ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA PEDAGOGICKÁ
KATEDRA VÝPOČETNÍ A DIDAKTICKÉ TECHNIKY

TRANSFORMACE ÚLOH Z PROGRAMOVÁNÍ 1 Z JAZYKU JAVASCRIPT DO JAZYKU PYTHON

BAKALÁŘSKÁ PRÁCE

Jan Plechatý
Učitelství pro základní školy, obor Učitelství informatiky pro základní školy

Vedoucí práce: Mgr. Filip Frank

Plzeň, 2023

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně s použitím uvedené literatury a zdrojů informací.

V Plzni, 27. června 2023

.....

vlastnoruční podpis

Poděkování

Především bych rád poděkoval Mgr. Filipu Frankovi za výbornou komunikaci, vstřícnost a užitečné rady během vypracovávání této práce. Zároveň bych chtěl poděkovat členům rodiny a blízkým přátelům za podporu nejen během vypracování této práce, ale i celého studia.

ABSTRAKT

TRANSFORMACE ÚLOH Z PROGRAMOVÁNÍ 1 Z JAZYKU JAVASCRIPT DO JAZYKU PYTHON

Tato bakalářská práce se zabývá transformováním úloh z jazyku JavaScript do jazyku Python. Práce obsahuje teoretickou část zabývající se základní teorií, výběrem prostředí pro zpracování úloh. V jednotlivých jazycích je věnována pozornost základní syntaxi.

V praktické části je popsána samotná transformace jednotlivých úloh. Úlohy psané v jazyce Python jsou vypracovány bez i s externími moduly. Pro zobrazování jednotlivých úloh je využíváno Tkinteru.

Klíčová slova

JavaScript, Python, HTML, CSS, Tkinter, datový typ, proměnná, funkce, parametr, argument, metoda, operátor, podmínka, cyklus, pole, list, element, widget, Canvas, modul

ABSTRACT

TRANSFORMATION OF TASKS FROM PROGRAMMING 1 FROM JAVASCRIPT TO PYTHON This bachelor's thesis deals with the transformation of tasks from JavaScript to Python. The thesis contains a theoretical part dealing with the basic theory, the selection of the environment for processing tasks. Attention is paid to the basic syntax in individual languages.

The transformation of individual tasks is described in the practical part. Tasks written in Python are developed with and without external modules. Tkinter is used to display individual Python tasks.

Keywords

JavaScript, Python, HTML, CSS, Tkinter, data type, variable, function, parameter, argument, method, operator, condition, loop, array, list, element, widget, Canvas, module

OBSAH

SEZNAM ZKRATEK

IDE – integrované vývojové prostředí

VS Code – Visual studio Code – vývojářské prostředí

JIT – Just-in-time – druh kompilace

JS – JavaScript – programovací jazyk

HTML – HyperTextMultipleLanguage – značkovací jazyk

CSS – Cascading Style Sheets – formátovací jazyk

ÚVOD

Tato práce se zaobírá transformací úloh z předmětu PGM1P z jazyku JavaScript do jazyku Python. V samotné práci je nejprve rozebrána obecná teorie týkající se vývojových prostředí a jejich využití. Dále nižší a vyšší programovací jazyk. Jako poslední nalezneme kapitoly týkající se kompilace, interpretace a JIT.

V teoretické části se zabýváme stručnou historií JavaScriptu a Pythonu. Následují kapitoly týkající se rysů, kde je rozebrána např. syntaxe, proměnná atd. V těchto podkapitolách se věnujeme využití a základního zápisu. Důležité je zmínit, že v kapitole JavaScript nalezneme podkapitoly popisující stručně jazyky HTML a CSS. Tyto jazyky úzce souvisí s jazykem JavaScript, kdy se tyto tři jazyky využívají pro vytváření webových stránek.

V praktické části jsou popsány úlohy JavaScript, tak i Pythonu z pohledu kódu. Úlohy napsané v jazyce Pythonu jsou popsány podrobněji, a to z důvodu, že transformace JS úloh, je hlavním cílem této práce. Popis úloh je strukturován podle vypracovávání na seminářích PGM1P.

1 SOUHRN ZÁKLADNÍ TEORIE A PŘÍPRAVA NA PRAKTICKOU ČÁST

V této kapitole je rozebírána samotná příprava na praktickou část. Jsou zde uvedeny příklady a stručný popis různých vývojových prostředí. Nacházejí se zde i podkapitoly týkající se základní teorie jako je vyšší a nižší jazyk nebo interpretace a kompilace jazyka.

1.1 VÝBĚR VÝVOJOVÉHO PROSTŘEDÍ

Před jakýmkoli programováním, musíme vybrat vývojové prostředí, ve kterém budeme celou práci vytvářet. Můžeme naleznout placené i neplacené verze. U placených verzí je standartně více funkcí a možností. U zpracovaných úkolů v této práci nepotřebujeme pokročilejší funkce IDE. V této kapitole se zaměříme na vývojová prostředí a textové editory, které lze využít v soukromém, školním i pracovním prostředí.

1.1.1 VISUAL STUDIO CODE

Visual Studio Code (VS code) je textový editor s open-source licencí. V tomto editoru můžeme zpracovávat a upravovat kód různých programovacích jazyků. Tento editor má po instalaci podporu Node.js, zahrnuje JavaScript i TypeScript. Jazyky jako například Python, musíme nainstalovat jako rozšíření. Visual Studio Code je jedna z oblíbených platforem z důvodu jednoduchosti, přehlednosti a rychlých operací a pro následující úlohy vhodným nástrojem. Toto vývojové prostředí je vhodné pro domácí, ale i pro školní použití. VS Code je podporován na různých operačních systémech, kdy oproti Visual studio s ním lze pracovat v operačním systému Linux. Ve VS code jsou vypracovány následné úlohy. (1) (2)

1.1.2 VISUAL STUDIO

Visual studio je integrované vývojové prostředí, které nabízí placenou i neplacenou verzi. Výhodou tohoto prostředí je práce na skupinových projektech nebo v pokročilých funkcích, které jsou například ve firmách často nutná. Mezi ně můžeme zahrnout grafické designery, kompilátory apod. Toto IDE se využívá i pro vývoj aplikací, kdy můžeme např. využít Windows Forms. Dále pomocí Visual Studia se vytvářejí počítačové hry a mobilní aplikace, kdy je možné nainstalovat rozšíření pro Unity. Visual Studio je vytvořeno pouze pro platformy Windows a macOS. (1)

1.1.3 PyCharm

I u tohoto vývojového prostředí nalezneme placenou a neplacenou verzi. Základní funkce PyCharm Community edition (neplacená verze) jsou editory kódu, debuggery a pluginy. Placená verze je na funkce bohatší. Nevýhodou neplacené verze je ta, že nenalezneme podpory například pro Node.js nebo React Native. Tudíž je vhodná pro základní práci v jazyce Python, ale pro širší použití je nutná placená verze. (3) (4)

1.1.4 Codepen.io

Codepen se řadí mezi sociální vývojové prostředí, které fungují přes webový prohlížeč. Po zaregistrování na stránce můžete vytvářet front-end kód. Je dobrý pro začátečníky. V tomto prostředí uživatel pouze zapisuje kód, zbytek obstarává codepen.io sám. Zároveň uživatel, žák, případně student získá okamžitou zpětnou vazbu. Pokud bychom chtěli v reálném čase někomu zobrazovat psaný kód, je zde ve zpoplatněné službě Profesorský režim. Ten by se mohl využít například při online výuce. (5) (6)

1.2 Jazyky nižší a vyšší úrovně

V následujících podkapitolách se věnujeme rozdělení jazyků na nižší a vyšší a jejich popisu. Zároveň uvádíme příklady jazyků v kontextu jejich zařazení.

1.2.1 Nižší jazyk

Nižší jazyk je charakteristický tím, že s ním dokáže pracovat samotný počítač (procesor). V těchto jazycích programátoři musí psát jednotlivé příkazy, které bude procesor zápis po zápisu provádět. Programátoři, by tak vytvářeli zdlouhavé zápisy a jejich fungování by bylo omezeno na procesor, pro který je zápis vytvářen. Můžeme sem zařadit například jazyk symbolických adres (Assembler), ten má slovní zkratky pro příkazy, které převádí do strojového kódu. (7) (8)

1.2.2 Vyšší jazyk

Jazyky vyšší úrovně jsou pro zápis instrukcí mnohem jednodušší. Vyplývají z logického zápisu, kterému rozumíme. Oproti nižším jazyku se zápis jednodušeji opravuje (debugging). Samozřejmě tento zápis potřebuje vyšší výkon pro zpracování kódu. Výhoda těchto jazyku spočívá v přenosnosti na různé typy zařízení, jednoduššího zápisu a možnosti využívat knihovny. Vyšší jazyk ke svému spuštění potřebuje interpreta nebo kompilátor. Mezi tento typ jazyka můžeme zařadit i JavaScript a Python. (8) (9)

1.3 INTERPRET VS KOMPILÁTOR

Oba tyto programy svým stylem přeměňují kód do podoby takové, aby jej mohl počítač číst a pracovat s ním. V dalších podkapitolách jsou zpracovány obecné využití interpreta a kompilátoru.

1.3.1 INTERPRET

Interpret překládá kód řádek po řádku do jazyka nižší úrovně. Interpretům analyzování kódu trvá podstatně méně času, než je tomu u kompilátoru, ale samotné provádění je pomalejší. Pokud při analyzování nalezne v kódu chybu, přestane a vypíše chybu v daném dokumentu. S tím souvisí i lepší ladění kódu. Interpretované jazyky jsou překládány za běhu programu. (10) (11)

1.3.2 KOMPILÁTOR

Kompilátor převádí zdrojový kód do nižšího jazyka, kdy ho nečte řádek po řádku jako interpret, ale nejdříve si ho přečte celý. Pokud je napsaný podle syntaxe, tak převedení proběhne v pořádku. Kompilátor je využíván pro soubory typu .exe. Tento typ souboru je jednou přeložen, to znamená vyšší rychlost provádění kódu. Pokud kód zkompilujeme je vytvořen jen pro konkrétní operační systém. (8) (11) (12)

1.3.3 JUST-IN-TIME

Just-in-time kompilace využívá výhod interpreta, tak i kompilátoru. Interpret překládá kód, který je zavolán. Probíhá zde k optimalizaci kódu, kdy při opakovaném volání kódu se jeho část nebo části vyhodnotí jako horká (hot). Pokud se tak stane, bude zkompilována. Tím je docíleno rychlejší vybavení kódu. Kompilace probíhá za běhu programu. (13) (14)

2 JAVASCRIPT

Na úvod je důležité zmínit co je to JavaScript. JavaScript je programovací jazyk, který se využívá u vytváření webu. Pomocí něho můžeme odkazovat a pracovat na HTML a CSS a vytvářet dynamický web. (15)

V této kapitole bude rozebrána teoretická i praktická část. V teoretické části je popsána krátká historie a základní syntaxe jazyka. Dále je zde věnovány kapitoly pro HTML a CSS, je to z důvodu toho, že zmíněné jazyky využívám v praktické části.

2.1 STRUČNÁ HISTORIE JAVASCRIPTU

Začátek tohoto jazyku začíná roku 1995 Brendanem Eichem, kdy byla vymyšlena první verze tohoto jazyka. Původně se jazyk měl jmenovat Mocha, poté se přejmenoval na LiveScript. Nedlouho poté vznikl finální název JavaScript byl roku 1997 standardizován. Standardizaci provedla standardizační organizace ECMA, která jej pojmenovala ECMAScript (ES1) a stal se standardem ECMA-262. (16) (17)

Nyní přeskočíme některé verze jazyka a přesuneme se do 21. století k novému prohlížeči Chrome, se kterou přišla společnost Google. Tento prohlížeč přinesl nový engine s názvem V8, který disponuje vysokou rychlostí pro načítání JavaScriptu. Implementuje zmíněný ECMAScript a Web Assembly, jenž se ve zkratce využívá pro rychlé spouštění kódu v prohlížečích. V roce 2009 byl vytvořen Node.js, který zahrnuje například zmíněný V8 engine, knihovny atd. Pomocí Node.js je možné spouštět JS i mimo prohlížeč. Hlavní výhodou je škálovatelnost, která se například využívá na serverech (back-endu). Engine V8 není jediný, například Firefox využívá SpiderMonkey. (18) (19) (20) (21)

2.2 RYSY JAZYKA JAVASCRIPT

V této kapitole je popsána základní syntaxi jazyka, datové typy. Dále jsou vysvětleny i z hlediska kódu proměnné, funkce, operátory, podmínky, cykly a pole.

2.2.1 ZÁKLADNÍ SYNTAXE

Syntaxe jako taková udává pravidla zápisů, vytváření jednotlivých částí kódu, kupříkladu funkcí, deklarace proměnných, cyklů atd. Za příkazy se píše středník, ale pokud se příkazy nacházejí na samostatném řádku, můžeme ho vynechat. Dále je důležité zmínit, že JS je case sensitive, to znamená že rozlišuje malá a velká písmena. Občas je nutné přidat ke kódu komentář nebo nějakou část kódu dočasně ignorovat. Pro jeden řádek komentujeme

dvěma lomítky, při zakomentování více řádků kódu před něj vkládáme lomítko s hvězdičkou a za něj hvězdičku s lomítkem (/* „blok kódu“ */). (22)

2.2.2 DATOVÉ TYPY

Datové typy jsou v jazyce pro rozlišování hodnot, což je důležité, jak pro práci se samotnými hodnotami, tak i čtení kódu. Mezi základní typy řadíme Number, String, Boolean, Undefined a Null. Jako první si popíšeme Number. Tento datový typ zastupuje čísla celá, ale i desetinná. Využívá se například pro výpočty. (22)

Dále zde máme String. Tento typ se využívá pro textové hodnoty. Pokud chceme hodnotě nastavit String, vložíme text nebo i číslo do uvozovek ('Zdravím tě!', '100'). Hodnoty textového typu můžeme pomocí znaku plus, když chceme například spojit dvě věty k sobě ('Ahoj, ' + 'jak se máš? ' = 'Ahoj, jak se máš? '). Nesmíme zapomínat, že i mezera se počítá jako znak. (22)

V programování je důležité rozeznat, kdy nastává „pravda“ (true) a „nepravda“ (false) situace. Tento datový typ se nazývá Boolean. Často jej využíváme u podmínek (if) nebo cyklů. Pomocí těchto dvou hodnot můžeme rozeznávat, zda například funkce, či proměnná nabyla jedné z hodnot. Například sčítáme čísla 2 a 3, a pokud se tyto hodnoty rovnají 5 (což je pravda), vypiš do paragrafu „pravda“, jinak „nepravda“. (22)

Undefined je hodnota, které například proměnná nabude ve chvíli, kdy nemá přiřazenou žádnou hodnotu. Oproti tomu hodnota Null reprezentuje žádnou hodnotu. Null můžeme přiřadit i proměnné jako dočasnou hodnotu, kterou máme v úmyslu později v kódu měnit. (23)

2.2.3 PROMĚNNÉ

Při vytváření si proměnná pro sebe ujme část paměti, do které ukládá hodnotu. JavaScript je dynamicky typovaný jazyk, to znamená, že se hodnotě ve stejně jmenované proměnné může měnit datový typ. Jsou tři typy proměnných, var, let a const. Var definujeme jako globální proměnnou, kterou je možné číst v celém dokumentu její deklarace. Za to let je lokální proměnná viditelná pouze v části kódu, ve které je vytvořena. To neznamená, že s hodnotou, které nabývá, nelze pracovat v jiných funkcích. Můžeme s ní pracovat jako s návratovou hodnotou funkce. Proměnnou typu const využíváme v případech, kdy hodnoty ji přiřazené nechceme měnit. Hodnotu do proměnné vkládáme přiřazením

(let cislo = 1), deklarování proměnné s názvem cislo a následné přiřazení hodnoty. Důležité je zmínit, že hodnoty v proměnných se můžou měnit. Proměnná nabývá poslední přiřazené hodnoty. (24) (25)

2.2.4 FUNKCE

Nyní se dostáváme k funkcím. Funkce jsou části kódu deklarovány klíčovým slovem *function*, následným přiřazením názvu a kulaté závorky, do kterých je možné přiřadit parametr funkce. Kód vkládáme do složených závorek, počáteční závorka začíná po kulatých závorkách. Pro spuštění bloku kódu funkce je nutné její zavolání. Volání jako takové se provádí tak, že napíšeme název funkce a za ní kulaté závorky, do kterých se uvádějí případné argumenty (*pozdrav ("ahoj“) {}*). Funkce mohou být volány i přes události (stisknutí tlačítka). Ještě nebylo zmíněno, co znamená argument funkce. Argument je hodnota, kterou je možné zadat funkci, která s ní pracuje jako s proměnnou. (26)

2.2.5 OPERÁTORY

Operátory se využívají pro provedení operací. Mezi základní operátory řadíme aritmetické, přiřazovací, porovnávací, řetězcové a logické. Pomocí aritmetických operátoru provádíme aritmetické operace, například sčítání (+), násobení (*) atd. Operátory přiřazení přiřazujeme hodnotu proměnným. Porovnávací operátory využíváme k porovnávání hodnot. Pro jedno rovnítko se používá přiřazování hodnot, proto ho jako operátora pro porovnání dvou hodnot nemůžeme použít. Z toho důvodu zde máme dvě základní porovnání, a to je dvě (==) nebo tři (===) rovnítka vedle sebe. První způsob využíváme pro porovnání hodnot, to znamená, že porovnáváme například hodnoty 1 (number) a "1" (string). Výsledek tohoto porovnání bude pravda (true). Ale, může nastat situace, kdy potřebujeme porovnávat i datový typ. Od toho je zde porovnání se třemi rovnítky. Při porovnání zmíněných hodnot třemi rovnítky, by výsledek byl false. Negace zapisujeme pomocí vykřičníku a dané porovnání (1 !> 2) nebo (1!===“2“). Menší nebo větší se v JS zapisuje stejně jako v matematice, a to pomocí špičatých závorek (>, <). Logické operátory AND a OR se využívají pro určení logiky mezi porovnávanými hodnotami. Operátor AND zapisujeme pomocí dvojice znaků ampersand (&&). Tento logický operátor se například používá v případě, kdy se musí splnit obě části podmínky, jinak vypíše hodnotu false.

Za to logický člen OR (||) vypíše hodnotu true i v případě, kdy se splní alespoň jedna z podmínek. (27)

2.2.6 PODMÍNKY

Podmínky využíváme v případě, kdy chceme provést kód za určitých požadavků. Definujeme ji slovem if(){}. Do kulatých závorek pak udáváme podmínku, která při splnění spustí kód ve složených závorkách. V kulatých závorkách můžeme mezi sebou porovnávat hodnoty se dotazovat, zda je něco pravda nebo nepravda. Pokud chceme, aby se provedl nějaký kód i v případě nesplnění podmínky, můžeme využít else{}. Kód vně složených závorek else se provede, při nesplnění podmínky předešlého if. Může nastat situace, kdy máme různé podmínky a chceme, aby se provedla pouze jedna z nich. To můžeme řešit pomocí else if(){}. Aby se spustil kód vně else if musí být splněna příslušná podmínka, jinak se neprovede daná akce. Dále máme možnost využít příkazu switch. Ten se vyhodnocuje za pomocí výrazu, který je porovnán s jednotlivými případy (case). Pokud se jeden z nich výrazu rovná, bude proveden daný blok kódu. (28) (29)

2.2.7 CYKLY

Cykly v programování používáme v situacích, kdy chceme docílit opakování části kódu. Cyklů máme několik typů. Jako první je cyklus for. For se skládá ze tří částí, kdy první je volitelná, ale většinou se zde uvádí proměnná, která se dále využívá v cyklu (např. pro procházení prvků pole). Ve druhé je uvedena podmínka, tou udáváme kolikrát cyklus proběhne. V poslední části uvádíme, co se stane po projetí cyklu. Tato část nemusí být uváděna. Jednotlivé části (výrazy) jsou odděleny středníkem. Dále můžeme mít situace, kdy potřebujeme opakovat kód, dokud je splněna určitá podmínka. Tento cyklus má název while(). Kód tohoto cyklu se opakuje do té doby, kdy vyhodnocení podmínky je true. Nevýhodou tohoto cyklu je jednoduché zacyklení kódu. Je zde i další varianta while cyklu, která se nazývá do while. Tento cyklus vždy alespoň jednou provede kód vně do a až poté kontroluje, zda byla splněna podmínka cyklu while. (30) (31)

2.2.8 POLE

Pole využíváme pro ukládání jedné nebo více hodnot. Samotné pole má pro každou hodnotu index, pomocí kterého je možné jednotlivé prvky procházet. Nesmíme zapomenout, že pole začíná s indexací od čísla 0. Pole v JavaScriptu je dynamické,

to znamená, že se jeho délka v průběhu kódu může měnit u přidávání hodnot do pole. Jednotlivé prvky můžeme například procházet pomocí cyklů. (32)

2.3 HTML

HTML (Hyper Text Markup Language) je hypertextový značkovací jazyk, který se využívá k tvorbě webových stránek. Vzhled HTML stránky vytváříme pomocí elementů, které mohou být párové a nepárové. Je důležité zmínit, že jednotlivé elementy mají atributy. Bez těchto atributů bychom například nemohli odkazovat na konkrétní element (*id*), přiřazovat hodnoty (*value=""*), vytvářet události (*onClick="funkce1()"*) atd. (33)

Nyní se dostáváme k jednotlivým částem HTML stránky. Skládá se ze čtyř částí. První je <!DOCTYPE>, část dokumentu, jenž poskytuje informace o verzi jazyku HTML (nyní nejčastěji HTML5). Jedná se o velice důležitou informaci pro prohlížeče, který díky tomuto zápisu „identifikuje", že se jedná o HTML stránku. Jako další zde máme hlavičku (*<head>*) dokumentu. Zde vytváříme odkazy k ostatním souborům, nastavujeme titulek stránky přes element *<title>* a při pokročilejší práci zde můžeme nastavovat i meta data (informace pro prohlížeče). Další částí je <body>. Pro nás nejpodstatnější z celého dokumentu. Vytváříme zde hrubý vzhled stránky, ve kterém vytváříme tagy (*<input>*). Je velice důležité si správně rozvrhnout strukturu a výsledný vzhled. (34) (35)

Jazyk HTML je zde využit především pro vytvoření kostry úloh a odkazování na jiné potřebné soubory (JS, CSS).

```
<!DOCTYPE html>
<html lang="cs">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="galerie.css">
    <script src="galerie.js"></script>
    <title>Galerie</title>
</head>
```

Obrázek 1 Ukázka zápisu DOCTYPE a head (Zdroj: vlastní)

```
<body>
    <div id="uvitani">
        <p class="text" style="color: violet; text-decoration:underline; text-decoration-color: red;">PGM1P </p>
        <h1 class="text" id="jmeno">Jan Plechatý</h1>
         <div id="tlacitka">
          <button onClick=obrazek1()>Obrázek1</button>
          <button onClick=obrazek2()>Obrázek2</button>
          <button onClick=zmena()>Změň</button>
         </div>
        <img id="obrazek"src="./obrazky_galerie/giphy.gif" alt="obrazek2">
      </div>

</body>
```

Obrázek 2 Ukázka zápisu elementů v body (Zdroj: vlastní)

2.4 CSS

CSS (Cascading Style Sheets) je formátovací jazyk, ve kterém se definují pravidla zobrazení například HTML stránek. Pro výběr prvků se využívají selektory. Mezi ně řadíme *inline* styly, *id*, třídy nebo můžeme nastavovat pravidla pro všechny nebo určité tagy. Je důležité zmínit, že máme různé priority selektorů. Nejvyšší má *inline* styl. Toto nastavení se přiřazuje ke konkrétnímu tagu v HTML dokumentu. Dále je zde *id*, což je jeden z atributů tagů. Nastavení CSS pravidel u *id* se zapisuje pomocí hashtagu a názvu *id* (*#nadpis* {}). Dále zde máme třídy, v CSS pravidlech se zapisují pomocí tečky a názvu třídy (*.text* {}). Jako další možnost přiřazení pravidel je ke všem tagům stejného názvu. Například můžeme přiřadit vlastnost písma všem paragrafům. Selektor tohoto typu zapisujeme pouze názvem daného tagu (*p* {}). Pokud chceme nastavit stejné pravidlo pro více různých tagů, například barvu pozadí, můžeme si je vložit do tagu * *. Pokud zvolíme tuto cestu, nastavíme pravidla pouze konkrétnímu divu, který v sobě má potomky, které zdědí jeho vlastnosti. Pro grafickou úpravu webového prostředí se v praxi využívají frameworky, soubor CSS pravidel (například Bootstrap). (36) (37) (38) (39)

Kaskádové styly jsou využity pouze u některých úloh, z důvodu toho, že nejsou hlavní částí daných úkolů. V této práci jsou zde pouze pro jednoduchou ukázku přiřazování stylů a vlastností jednotlivým tagům.

```
img{
  max-width:100%;
}
.text{
  font-family:Arial, Helvetica, sans-serif;
}
#jmeno{
  color: white;
}
```

Obrázek 3 Ukázka CSS (Zdroj: vlastní)

2.5 ÚLOHY

2.5.1 ÚLOHA 1.

Popis úlohy

Tato úloha je pojmenována Galerie. V této úloze jsou měněny obrázky podle stisknutého tlačítka. Je možné zde měnit obrázek na konkrétní, nebo kliknout na tlačítko změnit, které nám vykreslí jiný obrázek, než je tam doposud. Vzhled se skládá z nadpisů a podbarvení samotné galerie.

Pracujeme zde s HTML, CSS a JS. V samotném HTML je ukázána práce s hlavičkou (*head*) a tělem (*body*). Na potřebné soubory je zde odkazováno v hlavičce HTML a v těle (*body*) jsou vytvořeny elementy jako je *div*, *p*, *h1*, *button* a *img*. Tyto elementy mají i své události a atributy. Například zde můžeme zařadit událost *onClick*, která spustí funkci po kliknutí na tlačítko. Mezi atributy můžeme zařadit např. *id*.

Dále zde máme CSS soubor, ve kterém pro tagy s konkrétním *id*, třídou nebo na daném elementu nastavit úpravy. Jsou zde ukázky základních pravidel jako nastavení barvy, fontu, písma, šířky obrázku a nastavení pozice.

JavaScript je nejdůležitějším souborem z hlediska funkcionality. V této úloze jsou vytvořeny funkce, přes které voláme části kódu.

Popis kódu

Jako první věc v této úloze vytvoříme HTML a v hlavičce odkazujeme na soubory s JavaScriptem a CSS. V těle HTML dokumentu si vytvoříme *div*, kterému je přiděleno *id* pro nastavení barvy pozadí přes kaskádové styly. Jelikož chceme tuto barvu nastavit i pro další elementy, budeme je vytvářet vně *divu*. Vytváříme zde paragraf, nadpis *h1* a *div*, do kterého budeme vkládat tlačítka. Posledním elementem je *img*, pomocí kterého můžeme vkládat obrázek. U paragrafu vidíme dva možné selektory (třídu a styl), styl má vyšší prioritu, proto se nastaví na paragraf jeho pravidla. U nadpisu si můžeme všimnout podobného zápisu s rozdílem, že místo třídy je zde *id*. Nyní se dostáváme k tlačítkům, kde si můžeme všimnout události *onClick*, které přiřazujeme dané funkce. Jako poslední je zde obrázek, ke kterému je nastaveno *id* a pomocí atributu *src* přiřazujeme obrázek, konkrétně *giphy.gif*.

Kaskádové styly jsou zde definovány pro jednotlivé třídy, *id* nebo elementy s daným názvem. Jako příklad si můžeme uvést maximální šířku obrázku nacházející se v selektoru *img* nebo pravidlo pro tagy s třídou *text*, ve které je nastaveno *font-family*, jenž nám udává vzhled finálního písma, za pomoci několika podobných fontů.

Poslední částí je samotný JavaScript. Můžeme si všimnout funkcí s různými názvy. V první z nich odkazujeme na element *img*, jenž získáme pomocí *id*, a přes vlastnost *src* mu přiřazujeme *obrazek.jpg*.

```
document.getElementById("obrazek").src="./obrazky_galerie/obrazek.jpg";
```

Obrázek 4 Ukázka přiřazení obrázku (Zdroj: vlastní)

Ve druhé funkci přiřazujeme obrázek, kdy název obrázku je *giphy.gif*. V poslední funkci je nutné využít podmínky, ve které se dotazujeme na obrázek elementu *img*. Konkrétně se ptáme, zda se rovná *obrazek.jpg*, pokud ano, proběhne kód vně podmínky. Jinak řečeno změní se obrázek na *giphy.gif*. Jestliže původní podmínka se vyhodnotí jako nepravda, provede se kód vně *else*, měníme obrázek *img* na *obrazek.jpg*.

Obrázek 5 Výsledek Galerie JavaScript (Zdroj: vlastní)

2.5.2 ÚLOHA 2.

Popis úlohy

Druhá úloha ke nazvána Smajlíci. Skládá se z mnoha tlačítek a dvou paragrafů, některé jsou využívány pro vykreslení smajlíků a jiné pro práci s textem nacházejícím se v paragrafu. Po kliknutí na určité tlačítko budou vypsány dané smajlíky do paragrafů. Text prvního paragrafu se vždy změní podle posledního stisknutého tlačítka. Druhý paragraf se využívá pro výpis historie smajlíků, kam se vypisují všechny emotikony. Tato historie může být upravována, například mazána. Je zde i funkce, kdy do *inputu* můžeme napsat libovolný text a přes tlačítko ho vypsat do paragrafu.

Tato úloha se především zaměřuje na práci s proměnnými. Zároveň se zde pracuje i s novými elementy *input* a *p*. Je zde využíván pouze soubor HTML a JavaScript. V úloze se používá i parametr funkce. U funkcí *del*, *backspace*, *mazat* nebo *pridat* je využíváno podmínky. Dále je zde ukázka vyskakujících dialogových oken (*alert*).

Popis kódu

Nyní si rozebereme soubor HTML. Jsou zde vytvořená tlačítka. Každé z nich má událost *onClick*, které obsahují název funkce a kulaté závorky. V některých závorkách jsou parametry funkcí, jedná se o samotné smajlíky textové typu. S těmito parametry budeme dále pracovat v JavaScriptu. Můžeme si také všimnout, že u jednoho tlačítka je i *id*, jenž bude důležité pro získání obsahu *buttonu*.

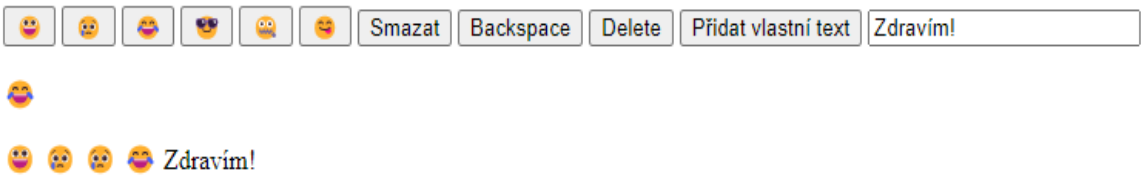
```
<button onClick="vesely()">😃</button>
<button id="smile"onClick="smutny()">😕</button>
<button onClick="vysmaty()">😂</button>
<button onClick="univerzalni('😍')">😍</button>
```

Obrázek 6 Ukázka tlačítek (Zdroj: vlastní)

Pro zadání hodnot od uživatele využíváme *input*, pomocí kterého dokážeme zpracovávat zadané hodnoty. Jediný atribut, který mu je zde přiřazen je *id*. Poslední dva elementy jsou paragrafy s příslušnými *id*, jenž zde používáme pro vypisování textových hodnot neboli smajlíků.

V JavaScriptu vidíme funkce, pomocí kterých vypisujeme smajlíky do paragrafů. Kód jednotlivých funkcích je různý, a to proto, abychom si ukázali různé možnosti zápisů. V první funkci je deklarovaná lokální proměnná, do které vkládáme hodnotu veselého smajlíku.

Vytvoříme si i proměnnou, do níž je uložen obsah elementu s příslušným *id*. Pomocí této proměnné jsme schopni vytvořit „historii“ smajlíků. Následně provedeme vypsání hodnot do paragrafů. Dostáváme se k další funkci s názvem *smutny*. Deklarujeme zde dvě proměnné, kdy do první přiřazujeme textový obsah *buttonu*, jinak řečeno text, který jsme u příslušného *buttonu* uvedli mezi otevírací a uzavírací tag. Do druhé proměnné přiřazujeme obsah paragrafu. Obě proměnné vypisujeme do paragrafů. Třetí funkce je velice jednoduchá, ale není úplně nejvhodnější. Přiřazujeme zde smějící smajlík staticky do elementů. Poslední funkce pro vypisování smajlíků obsahuje parametr, s nímž je pracováno jako s proměnnou a je vypisován do obou elementů. Na rozdíl od proměnné nemusíme parametr nijak deklarovat. Nyní se dostáváme k funkcím, které upravují vypsané hodnoty. Je zde funkce *mazat*, která nastaví prázdný text do obou paragrafů. Další funkcí je *backspace*, ve které si ukládáme obsah paragrafu s *id rada* do proměnné. V podmínce se dotazujeme, zda je prázdný, pokud je to pravda zobrazí se dialogové okno s daným textem. Jestliže nabývá alespoň jednoho znaku, tak přes metodu *substring* zkracujeme *text* o jeden znak. Proměnná je následně vypsána do paragrafu. Stejný postup je i u funkce *del*, kde jsme nastavili místo nuly číslo jedna. To znamená, že vynecháme první znak a následně vypíšeme zbytek řetězce. Poslední funkci používáme pro vkládání hodnot od uživatele do paragrafu. Vytvoříme proměnnou, do které přiřadíme hodnotu *inputu* přes vlastnost *value*. Je zde definována stejná podmínka jako u předchozích funkcí. Pokud je vyhodnocena jako nepravda, tak zobrazíme dialogové okno a pomocí operátoru plus zřetězíme text a hodnotu proměnné. Poslední nutností je vypsat proměnnou *rada* a zadaný text do paragrafu.



Obrázek 7 Výsledek Smajlíci JavaScript (Zdroj: vlastní)

2.5.3 ÚLOHA 3

Popis úlohy

Název úlohy je Diktát. Tato úloha má simulovat jednoduché vytváření diktátu, ve kterém po zavolání určitých funkcí přes tlačítka, nahrazujeme určité znaky (mě, s, y …) podtržítkem. V úloze je *textarea* pro psaní textu, který je dále „převzat“ do paragrafu. V něm pomocí tlačítek můžeme text upravovat. Dále jsou zde *buttony* pro mazání a zobrazení řešení. Znaky, za které byly nahrazeny podtržítka, můžeme přes tlačítko, vypsat původní text, jenž nám uživatel zadal se změněnou barvou a napsané velkými písmeny.

Popis kódu

Stránku HTML si rozdělíme do dvou *divů*. V prvním vytváříme nadpis a *textarea*, do kterého budeme zadávat text. V druhém definujeme *butony* s jejich událostmi *onClick* a paragraf, do kterého budeme vypisovat text, jenž budeme moci upravovat pomocí tlačítek.

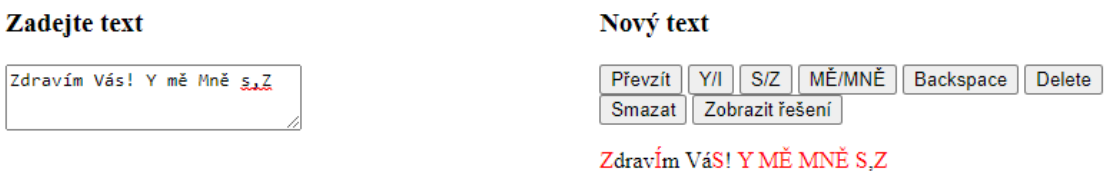
V kaskádových stylech je uvedena třída s názvem *tvary*. V ní využíváme vlastnost *float* s hodnotou *left*. Díky tomuto zápisu docílíme zarovnání *divů* vedle sebe. Dále jim nastavujeme šířku 40 % obrazovky a *margin* s hodnotou 25 pixelů. *Margin* využíváme pro vytvoření prostoru kolem elementu o zadanou hodnotu.

První funkci v JS využíváme pro převzetí kódu. Již v této části si musíme text, který nám byl od uživatele zadán uložit do globální proměnné. Deklarujeme ji na prvním řádku pomocí klíčového slova *var*. Zároveň si text uložíme i do lokální proměnné ve funkci *prevzit*, ve které vytváříme větvenou podmínku. V ní se ptáme na různé délky textu a pro každou možnost voláme metodu *alert* (dialogové okno). Poté text zadaný od uživatele vložíme do paragrafu s *id diktat*. Dále vytvoříme několik funkcí, pomocí kterých budeme nahrazovat určité znaky za podtržítka. Jako první krok musíme deklarovat proměnnou, do které přiřazujeme obsah paragrafu. Následuje podmínka, v níž jsou všechny znaky převedené na malé a pomocí metody *indexOf* se dotazujeme, zda se znak v textu nachází. Vidíme, že pomocí operátoru OR (||) je možné se dotazovat na různé znaky, s tím že pokud se alespoň jedna podmínka vyhodnotí jako pravda, tak se provede nahrazení textu.

```
if ((text.toLowerCase().indexOf("s") > -1)||(text.indexOf("S") > -1)||(tex
  {
    text=text.replace(/s/ig,"_");
```

Obrázek 8 Ukázka podmínky a nahrazování znaku

U funkce *memne* nevyužíváme převodu na malá písmena. Pro nahrazení znaků využíváme metody *replace*, která nabývá dvou parametrů, kdy do prvního určujeme znaky (regulární výraz), které chceme nahradit. Za lomítkem jsou příznaky (*i* a *g*). Pomocí malého *i*, jsme schopni ignorovat velikost znaku a malé *g* nám zaručuje globální nahrazení znaků. Upravený text vypíšeme. Dále zde máme funkce *backspace*, *del* a *smazat*. Jejich kód jsme si již popsali v předešlé úloze. Jediný rozdíl zde nastává ve funkci *smazat*, kdy po splnění dané podmínky se nám zobrazí dialogové okno s dvěma tlačítky. Po kliknutí na *OK* se text smaže, jinak ne. U příslušných možností se zobrazí dialogové okno, které nás informuje, co se s „diktátem“ stane. Poslední funkce je *reseni*, ve které se dotazujeme, zda text v paragrafu není stejný jako v globální proměnné. Pokud je to pravda, využíváme metody *replace* a regulárních výrazů, tentokrát na původním textu. Znaky nahrazující podtržítka jsou velkými písmeny a červenou barvou.



Obrázek 9 Výsledek Diktát JavaScript (Zdroj: vlastní)

2.5.4 ÚLOHA 4

Popis úlohy

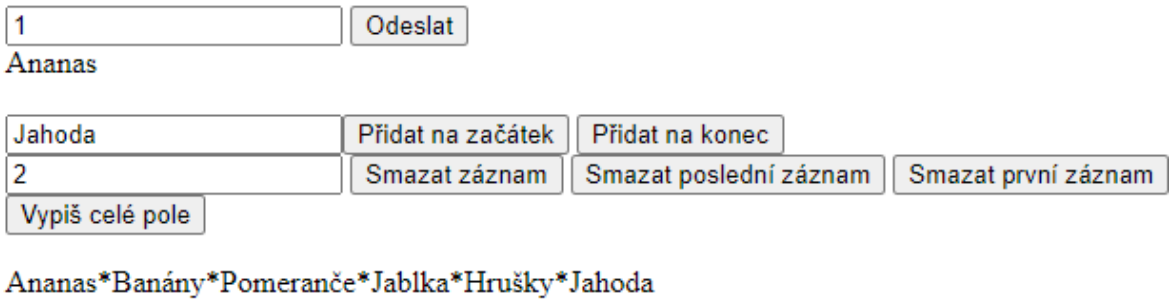
Úloha Ovocné hrátky obsahuje tři *inputy*, sedm *buttonů*, paragraf a *div*, do kterých budeme vypisovat. V úloze pracujeme s polem (*array*). Nejprve je v této úloze ukázáno vytvoření pole, jenž je uloženo do globální proměnné. V úloze pracujeme s pozicemi prvků pole, například u vypisování prvku nebo jeho mazáním na námi zadané pozici. Z pole můžeme odebírat prvky i přes tlačítka pro smazání prvního nebo posledního záznamu. Dále je možné na začátek nebo nakonec pole přidávat zadané hodnoty z inputu. Jako poslední práci s polem si zde můžeme všimnout vypisování jednotlivých hodnot uložených v poli do paragrafu.

Popis kódu

V dokumentu HTML definujeme *inputy*, tlačítka, paragraf a *div*. U *inputu* je možné vidět atributy *value* a *placeholder*. Rozdíly jsou na první pohled patrné, *value* je nastavená

hodnota daného elementu. Chceme-li hodnotu změnit musíme ji z inputu smazat ručně. Hodnota *placeholderu* se po zadání hodnoty od uživatele smaže.

V JavaScriptu je deklarované pole, ve kterém jsou názvy ovoce. Jsou zde funkce, které různě pracují s daným polem. Jako první je funkce, která vypisuje hodnoty z pole do paragrafu a mezi jednotlivé hodnoty vkládá hvězdičku. Další funkce má za úkol podle zadaného čísla do prvního inputu vypsat ovoce z pole. Je nutné řešit i logické úvahy uživatele, když bude chtít vypsat první ovoce, tak do inputu nebude zadávat číslo nula, ale zadá nám pravděpodobně číslo jedna. Pro pole je ovšem 1 druhá pozice, proto je nutné vstup o jednu snížit. Dále je zde funkce, díky které je možné vkládat hodnotu na poslední pozici v poli. K tomu je využívána hodnota délky pole. Je zde i volána funkce *vypis*. V další funkci řešíme vkládání hodnoty na první pozici pomocí metody *unshift*. U funkcí pro mazání prvního a posledního se pouze mění použité metody na poli *ovoce*. Oproti tomu mazání hodnoty na libovolné pozici musíme od vstupu odečíst číslo jedna. Pro smazání libovolného prvku z pole využíváme metody *splice*, ve které určujeme, kde chceme začít s mazáním a kolik znaků smažeme.



Obrázek 10 Výsledek Ovocné hrátky JavaScript (Zdroj: vlastní)

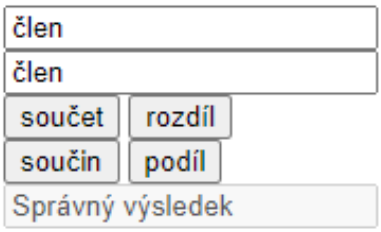
2.5.5 ÚLOHA 5

Popis úlohy

Kalkulačka. Úloha je zaměřená na práci s datovým typem *number*. Obsahuje tři *inputy*, čtyři tlačítka. *Inputy* využíváme jako vstup čísel od uživatele, s kterými se dále pracuje. Po zadání číselných hodnot přejdeme k tlačítkům. Každý *button* znázorňuje jednoduchou matematickou operaci (součet, rozdíl, násobení, dělení). Po kliknutí na některé z tlačítek se provede mezi dvěma zadanými čísly konkrétní operace. Výsledek tohoto výpočtu se vypíše do *inputu*.

Popis kódu

V této úloze jsou tlačítka s událostí *onClick* a *inputy*, kde jeden z nich má atribut *disabled*, nastavený jako *true*, proto do něj nelze vkládat hodnoty. Všechny funkce pro matematické operace mají stejný blok kódu, ve kterých se pouze mění znaménko u výpočtů. V jednotlivých funkcích deklarujeme lokální proměnné, do kterých přiřazujeme hodnotu zadanou od uživatele a pomocí podmínek se dotazujeme, zda jsou to čísla. Pokud ano, musíme provést přetypování na datový typ *number*. Pokud bychom přetypování neprovedli nemohli bychom provádět matematické operace. Do posledního inputu vypisujeme výsledek dané výpočetní operace mezi dvěma členy. Jediný rozdíl nastává ve funkci *podil*, kde uvádíme podmínku, kdy dělitel nesmí být nula. Zároveň je zde pomocí metody *toFixed* výsledek zaokrouhlován na dvě desetinná místa. Můžeme si všimnout, že v každé funkci je volána funkce s názvem *NeniCislo*. Ta na základě špatně zadaných hodnot do inputu, je schopná rozeznat, ve kterém nebylo zadáno číslo. Pro zjištění chyby využíváme podmínek. Pro vypsání chyb je využíváno dialogových oken.



Obrázek 11 Výsledek Kalkulačka JavaScript (Zdroj: vlastní)

2.5.6 ÚLOHA 6

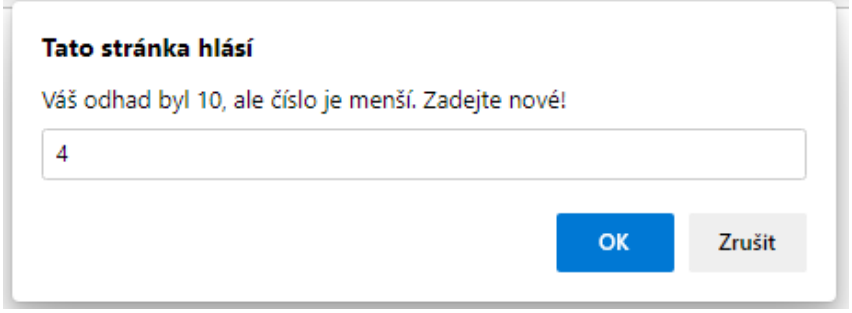
Popis úlohy

Tato úloha je pojmenována Hádání čísel. Úloha spočívá v zadání čísel do dvou *inputů*, kdy první reprezentuje začátek a druhé konec intervalu. V intervalu se vybere náhodné číslo a uživatel, se jej pokusí uhádnout. Pro hádání čísla je zde využíváno dialogového okna obsahující *input* (*prompt*). Uživatele se dotazujeme na tip do té doby, dokud neklikne tlačítko zrušit (nechce hrát) nebo při uhádnutí čísla. Pro opakované dotazování se využívá cyklu *while*.

Popis kódu

V HTML využíváme dvě úrovně nadpisů, tři *inputy* a dvě tlačítka, které mají události *onClick*. V samotném JS je deklarovaná proměnná typu *var*. V první funkci deklarujeme dvě proměnné, do kterých přiřazujeme hodnoty *inputů*. Dále se využívá podmínky, která nám vyhodnocuje, zda oba *inputy* jsou čísla. Pokud je to vyhodnoceno jako pravda, tak musíme provést přetypování na datový typ *number*. Do samotné globální proměnné je přiřazena hodnota, kterou získáme za využití metod objektu *Math*. Pomocí těchto metod docílíme zaokrouhlení na celá čísla a vygenerování čísla, které uživatel bude hádat. Nyní si popíšeme další funkci, ve které jsou deklarované proměnné. Nejprve se ptáme, zda tip od uživatele je číslo, pokud ano je nutné přetypovat číslo na *number*. Pomocí klíčového slova *while* vytvoříme cyklus, který bude probíhat, dokud bude splněna jeho podmínka (tip se nerovná výsledku). V samotném cyklu jsou další podmínky, pomocí kterých jsou řešeny situace, kdy nový odhad je menší nebo větší než výsledek, případně jestli nabývá hodnoty *null*. Pokud jsme výsledek neuhádli, tak se nám zobrazí *prompt* s informací, zda náš tip je větší nebo menší než výsledek a uživatel má možnost zadat nový tip. Hádání čísla je možné zrušit, a to za pomoci stejnojmenného tlačítka. Pokud se tip rovná výsledku, zobrazí se dialogové okno, ve kterém je napsáno počet pokusů, než bylo číslo uhodnuto. Pokud tip nebyl správný a chceme zrušit hádání čísla, musíme pro ukončení cyklu do proměnné *odhad* přiřadit hodnotu *vysledek* a nastavíme *vyhru* na *false* pro zobrazení daného dialogového okna.

Obrázek 12 Výsledek Hádání čísel JavaScript (Zdroj: vlastní)



Obrázek 13 Prompt (Zdroj: vlastní)

2.5.7 ÚLOHA 7

Popis úlohy

Úloha zaměřena na práci s objektem a nastavení jeho vlastností, kterým jsou vyplňovány hodnoty. Objekt je i s nimi uložen na určitou pozici pole. V této úloze budeme jednotlivé hodnoty objektů zřetězené do *stringu* vypisovat do paragrafu, či *alertu*. Přidáním auta do pole se vlastnosti jednotlivých objektů vypíšou do paragrafu. Na každý řádek připadá jedno auto. Dále v úkolu máme tlačítko souhrn a celková hmotnost. Po kliknutí na souhrn se nám zobrazí *alert*, ve kterém budou vypsána všechna auta a celková hmotnost (hmotnost sečtena za všechny auta). Celkovou hmotnost si můžeme zobrazit i v inputu po kliknutí na stejnojmenné tlačítko.

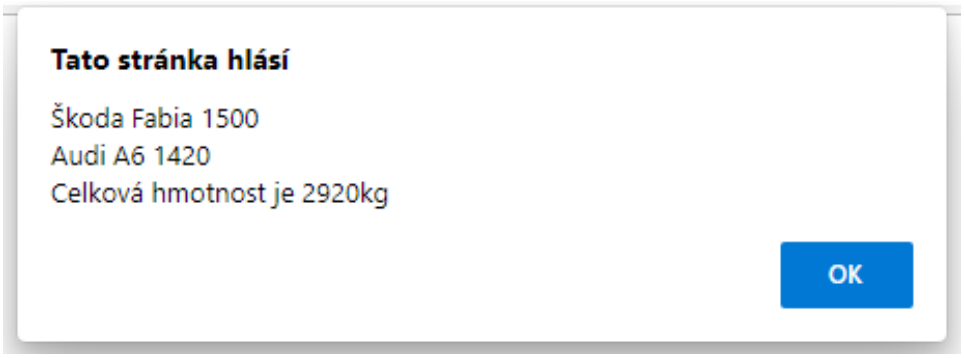
Popis kódu

V HTML jsou vytvořeny čtyři *inputy*, tři tlačítka a paragraf. Jednotlivá tlačítka mají události *onClick* a ostatní elementy mají přiřazený atribut *id*.

V samotném JS je vytvořené prázdné pole, do kterého budou postupně vkládány objekty. Tyto objekty vytváříme ve druhé funkci pomocí jednoduchého zápisu. Objektu auto přiřazujeme vlastnosti *znacka*, *model* a *hmotnost*. Dále v kódu musíme upřesnit za jakých podmínek kód bude pokračovat. To určujeme pomocí podmínek, pokud jsou splněny, tak do jednotlivých vlastností objektu jsou přiřazeny hodnoty, které nám uživatel zadal do inputů. Následně je dané auto vloženo na poslední pozici pole. Jakmile začneme vkládat více objektů, z pole se stává dvourozměrné pole. Abychom mohli přistupovat k jednotlivým objektům i jejich vlastnostem odděleně. Na poslední řádce funkce vidíme volanou funkci *vypsat*. Pomocí této funkce přes cyklus *for* vypisujeme všechny hodnoty vlastností jednotlivých objektů, které se v poli nachází. Vypisujeme je na nový řádek za využití elementu *br*. Další funkce se zabývá počítáním celkové hmotnosti. Využíváme cyklus *for*, je

to z důvodu dobrého využití indexu neboli procházení jednotlivých hmotností objektů v poli. Samotnou hodnotu je nutné přetypovat na typ *number*, protože provádíme matematickou operaci. Následuje *return*, který zde používáme pro návrat hodnoty hmotnost pro *souhrn*. Samotná funkce *souhrn* má rozdílný zápis než předešlé úlohy. V celé funkci je vytvořen pouze *alert*, do kterého jsme vložili návratovou hodnotu funkce *vypsat*, v níž nahrazujeme *br* za *\n*. Je to z důvodu, že pro odřádkování v dialogovém oknu se využívá *\n*. Dále v *alertu* vypisujeme text a hodnotu celkové hmotnosti.

Obrázek 14 Výsledek Auta JavaScript (Zdroj: vlastní)



Obrázek 15 Souhrn (Zdroj: vlastní)

2.5.8 ÚLOHA 8

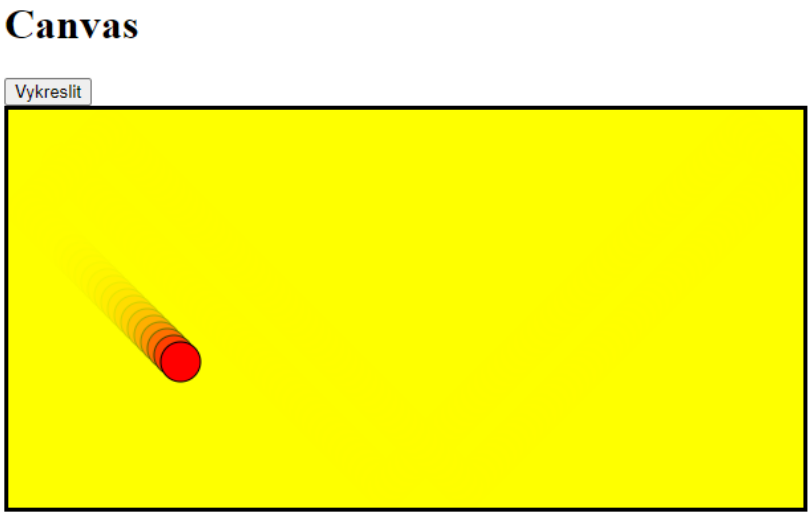
Popis úlohy

Úloha pojednává o vykreslování geometrického útvaru koule v c*anvasu*. Dále je ukázána práce s globálními proměnnými, kdy se jejich hodnoty různě mění za určitých podmínek. Po kliknutí na tlačítko se koule „rozpohybuje“, kdy při „dotyku“ okraje *canvasu* se odrazí pod inverzním úhlem.

Popis kódu

V této úloze v těle HTML je pouze element *canvas*, který má atributy *id*, šířku a výšku. V CSS nastavujeme přes selektor *id platno* barvu a okraje elementu *canvas*. V JS ve funkci *vykreslit* řešíme podmínky, zda jednotlivé pozice koule je menší nebo větší daným hodnotám reprezentující hranice *canvasu*. Proměnné *x* a *y* reprezentují pozici samotné koule a *dx* a *dy*

jsou hodnoty, které určují o kolik se má daný útvar posunout. Pokud se pozice vykresleného geometrického útvaru nerovná hranici *canvasu*, tak k inverzi proměnných *dx* a *dy* nedochází. Jestliže dosáhne pozice hranice *canvasu*, tak se hodnoty invertují a tím se změní směr vykreslování koule. Následuje vykreslení samotného koule. K tomu využíváme metody, kterými vykreslujeme samotný geometrický útvar v *canvasu*. V metodě *arc*, je možné si všimnout, že do parametrů přiřazujeme proměnné *x* a *y*. Pro nastavení barvy využíváme vlastnost *fillStyle*. Pokud chceme kuličku vykreslovat v určitých intervalech můžeme použít metodu *setInterval*, kdy do prvního parametru uvádíme samotnou funkci *vykreslit* a do druhého parametru uvádíme za kolik milisekund se funkce znovu spustí.



Obrázek 16 Výsledek Kulička JavaScript (Zdroj: vlastní)

2.5.9 ÚLOHA 9

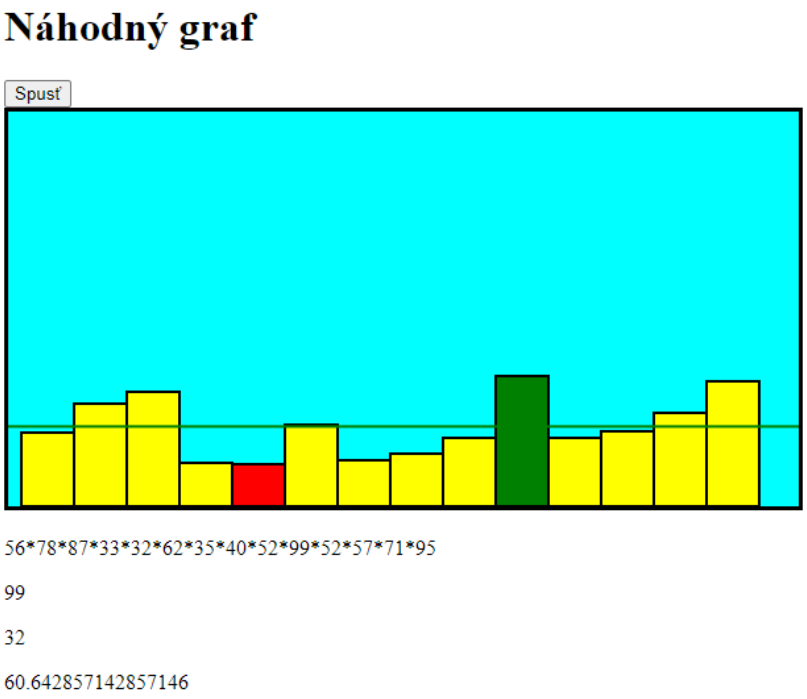
Popis úlohy

V této úloze jsou vykreslovány sloupečky, znázorňující sloupcový graf, kdy jejich hodnoty jsou dány hodnotami v poli. V těle HTML využíváme element *canvas*, dále tlačítko a paragrafy, do kterých vypisujeme hodnoty (pole, maximální, minimální a průměr). Pomocí CSS podbarvujeme a vytvoříme ohraničení *canvasu*. V JS budeme pracovat s proměnnými, podmínkami, cykly, a geometrickými útvarem obdélník. Pro znázornění průměru v grafu je vykreslena linie. Důležitá je zde práce s podmínkami, určující přiřazování barvy obdélníku.

Popis kódu

V těle HTML je tlačítko, kterým zavoláme funkci *vykreslit*. Dále je zde element *canvas*, do kterého budeme grafy vykreslovat. Následují čtyři paragrafy, které využijeme pro vypisování všech hodnot v poli, z kterých vypočítáme a vypíšeme aritmetický průměr. Ostatní paragrafy využíváme pro vypsání největší a nejmenší hodnoty.

V JavaScriptu se nachází pouze jediná funkce s názvem *vykreslit*. Nejprve si deklarujeme proměnnou *obsah*, do které přiřadíme 2D kontext elementu *canvas*. Dále definujeme prázdné pole. Deklarované proměnné jsou lokální, protože je využíváme pouze v jedné funkci, to neznamená, že by nemohly být globální, ale v tomto případu je to zbytečné. Dále smažeme vše, co je nakresleno v *canvasu*. Přes cyklus postupně přidáváme náhodné hodnoty do pole (20 až 100). Jakmile jsou hodnoty v poli můžeme provést výpisy do paragrafů. Následně je nutné vykreslit jednotlivé sloupečky podle hodnot v poli. Je zde deklarovaná proměnná *x*, pomocí které určujeme pozici sloupečků. Pokud se hodnota na dané pozici pole rovná největší, tak se sloupec podbarví zeleně, jestliže je nejmenší červeně, jinak žlutě. Můžeme si všimnout, že v samotném vykreslování musíme řešit posouvání sloupců od sebe (přičítání do proměnné *x*). Pro vykreslení výšky (*y*) je nutné zadat hodnotu z pole na pozici *i* v inverzním tvaru, abychom vykreslovali sloupce nahoru, a ne pod úroveň *canvasu*. Jako poslední vykreslování řešíme znázornění průměru, a to pomocí linie. Předtím je důležité vypočítat aritmetický průměr, který vypisujeme do paragrafu. Linii nastavíme na minimální a maximální šířku a od výšky odečteme průměr. Následně vykreslíme linii.



Obrázek 17 Výsledek Grafy JavaScript (Zdroj: vlastní)

3 PYTHON

Python je řazen mezi interpretované vysokoúrovňové jazyky a je využíván například pro vývoj aplikací. (40)

V kapitole Python je zabýváno teoretickou a praktickou částí. V teoretické se nachází stručná historie, rysy a v krátkosti virtuálnímu prostředí a modulům z PyPi. V praktické části jsou rozebrány transformované úlohy.

3.1 STRUČNÁ HISTORIE PYTHONU

Jazyk byl navrhnut Guido van Rossumem v roce 1991. O tři roky déle byla vydána první verze jazyka Python. Důvodem jeho vzniku byla myšlenka vytvořit programátorský jazyk, jenž bude lépe čitelný a jeho zápis bude zkrácen oproti jiným jazykům. Python je inspirován jazykem ABC. Samotný Python si prošel i mnohými verzemi, v nich získával postupně nové funkční nástroje a funkce. (41) (42)

Nyní si porovnáme některé z rozdílů mezi druhou a třetí verzí Pythonu. Ve třetí verzi oproti druhé došlo ke změně syntaxe, aby se předešlo redundanci, případnému opakování kódu. Mezi další výhody můžeme zařadit zajištění vyšší bezpečnosti a lepší čitelnosti kódu. Dále je zde vylepšená podpora s Unicode, kdy Python 2 využívá ASCII, ale Python 3 je nastaven na práci s Unicode. Důležité je zmínit, Python 3 je zpětně nekompatibilní. (43)

3.2 RYSY JAZYKA PYTHON

V následujících podkapitolách je popsána základní syntaxe jazyka a datové typy. Dále jsou popsány proměnné, funkce, operátory, podmínky, cykly a listy spolu se slovníkem.

3.2.1 ZÁKLADNÍ SYNTAXE

Syntaxe jazyka Python se vyznačuje svou přehledností. Jeho syntaxi můžeme prohlásit za čistou syntaxi, tomu napomáhá odsazení. Je nutné dodat, že části kódu, které jsou na stejné „úrovni“ ku příkladu if a else, budou mít stejné odsazení. Dále je důležité zmínit ukončení řádku kódu. V Pythonu se žádným konkrétním znakem řádek neukončuje. Pokud bychom chtěli řádek zalomit a navázat na zápis z předchozího řádku je možné využít znaku zpětného lomítka. V některých případech se nám mohou vyplatit komentáře kódu. Pro zakomentování jednoho řádku využíváme znaku hashtag (#). Pokud bychom chtěli komentovat část kódu, musíme před a za něj umístit tři apostrofy (’ ’ ’). (44)

3.2.2 DATOVÉ TYPY

Mezi základní datové typy v Pythonu můžeme zařadit str, int, float, list, bool a NoneType. Pro textové hodnoty je datový typ str. Hodnoty tohoto datového typu jsou vkládány mezi uvozovky. Využívá se například pro vypisování hodnot textové řetězce. Pro celočíselné hodnoty je int. Pokud bychom potřebovali pracovat s číselnými hodnotami s desetinnou čárkou využili bychom datového typu float. Pro list je popsána samotná podkapitola. Dále je zde datový typ bool neboli True a False. Posledním datovým typem je NoneType, ten se například využívá pro nastavení hodnoty proměnné, kdy ji nastavujeme nulovou hodnotu. (45)

3.2.3 PROMĚNNÉ

Proměnné v Pythonu nemají pro své vytvoření klíčové slovo. Samotná proměnná vznikne názvem a přiřazením (=) hodnoty. Python je dynamicky typovaný jazyk, to znamená, že se datový typ hodnoty proměnné může měnit. Je možné od sebe odlišit lokální a globální proměnnou. Lokální proměnná se deklaruje například vně funkce, kdy je viditelná jen v daném části kódu. Důležité je zmínit, že může mít stejný název jako lokální proměnné v jiných funkcích nebo jako globální proměnná. Pokud bychom použili stejný název jako u některé z globálních proměnných nesmíme globální proměnnou v bloku kódu využít. Samotné definování globální proměnné například ve funkci se provádí přes klíčové slovo global. Tím docílíme možné práci s globální proměnnou. (46) (47) (48)

3.2.4 FUNKCE

Funkce vytváříme klíčovým slovem def, kdy do funkce vkládáme blok kódu, který při jejím volání bude proveden. Po def následuje název funkce a kulaté závorky, do kterých je možné uvést parametr. Za kulatými závorkami uvádíme dvojtečku. Funkce mohou být například volány v jiných částech kódu nebo přes tlačítko. (49)

3.2.5 OPERÁTORY

Operátory využíváme pro provedení operací mezi hodnotami. Mezi základní operátory můžeme zařadit aritmetické, porovnávací, přiřazení, logické, identity a členství. Aritmetické operátory využíváme například pro sčítání, odčítání násobení atd. Pro porovnání různých hodnot se používají operátory porovnávací. Díky nim můžeme porovnávat, zda jsou hodnoty stejné, větší, menší a jiné základní porovnání. Pokud potřebujeme dotaz znegovat využíváme vykřičník. Přes operátory přiřazení přiřazujeme hodnoty například proměnným.

Logické můžeme využít u podmínek, kdy jich musí být splněna jedna nebo více. Řadíme mezi ně and, or a not a po logickém vyhodnocení vracejí True nebo False. Pomocí not můžeme např. negovat části nebo celou podmínku. Mezi operátory identity jsou řazeny is a is not. Nimi se ptáme, zda jsou nebo nejsou porovnávané hodnoty stejné i z hlediska paměti. Můžeme je využít například pro porovnání, zda je proměnná True. Posledními operátory jsou in a not in. Využívají se v situacích, kdy chceme zjistit, zda se nějaká hodnota nachází například v listu. Výsledkem je True nebo False. (50)

3.2.6 PODMÍNKY

Podmínky používáme, kdy chceme spustit část kódu při jejím splnění. V podmínkách se často využívají operátory. Samotnou podmínku vytváříme přes klíčové slovo if, kdy na rozdíl od JavaScriptu nemusíme podmínku uvádět do kulatých závorek. Na konci podmínky vkládáme dvojtečku. Blok kódu, jenž chceme spustit při splnění podmínky nevkládáme do složených závorek, ale pomocí tabulátoru kód odsadíme. Podmínku můžeme větvit, kdy využíváme else nebo elif. Else se provede v případě nesplnění předchozí podmínky. Dále je tu elif, které se využívá při nesplnění podmínky předchozí s tím, že se dotazujeme na podmínku další. (51)

3.2.7 CYKLY

Cykly využíváme pro opakování určitého bloku kódu. Jako první si popíšeme cyklus while. Tento typ cyklu využíváme tehdy, kdy chceme docílit opakování bloku po dobu splnění podmínky. Podmínku cyklu nemusíme uvádět do kulatých závorek a na jejím konci je dvojtečka. Blok kódu vně cyklu musí být odsazen. Dále zde máme cyklus for. Ten využíváme pro procházení jednotlivých prvků listu, řetězce atd. Pro procházení se využívá proměnné operátoru in. Podmínky pro různá porovnání se uvádějí uvnitř cyklu. (52) (53)

3.2.8 LIST A SLOVNÍK

List se využívá pro ukládání hodnot s různým datovým typem. Hodnoty mohou být ukládány do listu postupně. List deklarujeme přes hranaté závorky, dovnitř můžeme vložit hodnoty. Pro zobrazování prvků z listu můžeme využít různé způsoby. Zároveň je zde možný jednoduchý výpis části listu, za využití dvojtečky. Různé práce s listem jsou ukázány v úkolech v praktické části. Nyní se dostáváme ke slovníkům. Slovníky jsou seznamy, které jsou charakteristické tím, že jejich položky jsou charakterizovány v párech. Samotný pár je

tvořen za pomoci klíče a hodnoty, kdy hodnotu položky slovníku jsme schopni získat přes název klíče. Ve slovníku není možné jednomu klíči přiřadit více hodnot. (54) (55)

3.3 MODULY PYPI

PyPi (Python Package Index) je uložiště pro balíčky Pythonu, ke kterým je bezplatný přístup. Pomocí těchto balíčků můžeme využít moduly, funkce, příkazy atd. Pro instalaci balíčků potřebujeme mít nainstalovaný nástroj pip, přes který jsme schopni instalovat jednotlivé balíčky. Výhoda modulů z PyPi je v před připravenosti struktur kódu, kterými si můžeme ušetřit mnoho práce. (56) (57)

3.4 VIRTUÁLNÍ PROSTŘEDÍ

Virtuální prostředí je struktura složek, která se využívá pro izolaci jednotlivých projektů. Tyto výhody nastávají, kdy pro různé projekty potřebujeme různé moduly nebo odlišné verze Pythonu. Pro stahování knihoven atd. do virtuálního prostředí se běžně používá příkaz activate. Prostředí je možné i deaktivovat, a to příkazem deactivate. Výhodou virtuálního prostředí je i přenositelnost na jiná zařízení s nainstalovanými balíčky atd. (58)

3.5 ÚLOHY BEZ EXTERNÍCH BALÍČKŮ

Všechny úlohy byly vytvořeny pomocí grafického modulu tkinter. Tento modul byl zvolen pro vytvoření podobného vzhledu úloh jako v JS. Samotný tkinter obsahuje různé widgety, jako jsou tlačítka, vstup pro uživatele atd. Tkinter je zabudován v Pythonu, a proto se neřadí mezi externí moduly. Jednotlivé úlohy budou popisovány ve stejném pořadí jako tomu bylo u JavaScriptu.

U každé úlohy je v první řadě důležité importovat samotný *tkinter*. Dále mu můžeme nastavit zkratku pomocí klíčového slova *as* a stanovíme název zkratky. Využívá se pro lepší čitelnost a psaní kódu. Další řádek kódu, který je v některých úlohách využíván je importování modulu z *tkinteru*, například *messagebox*. Je možné importovat z tkinteru vše pomocí znaku hvězdičky (*). Další standartním kódem je ukládání instance třídy *Tk* (hlavního okna aplikace) do proměnné s názvem *root*. Hlavnímu oknu můžeme přiřazovat jednotlivé widgety (tlačítka, vstup …). Následně oknu určujeme jeho šířku a výšku za pomocí metody *geometry*, která se volá na *root*. Hlavnímu oknu můžeme přiřadit i titulek. K tomu využíváme metodu *title*, do které přiřadíme textovou hodnotu názvu.

```
import tkinter as tk
root=tk.Tk()
root.geometry("1000x600")
root.title("Galerie")
```

Obrázek 18 Ukázka importování tkinteru a nastavení hlavního okna (Zdroj: vlastní)

3.5.1 ÚLOHA 1

Popis úlohy

Tato úloha je modifikací Galerie z JavaScriptu. rozdělena do dvou částí. První z nich je část tvořená z textu, popisků a tlačítek. Druhá je tvořena ze samotného obrázku. V tomto úkolu je ukázáno importování *tkinteru*, vytvoření a vykreslení hlavního okna, vytvoření widgetů (*Frame*, *Label*, *Button*). Widgetům je přiřazován rodič, argumenty a metody, pomocí kterých můžeme využít různé možnosti umístění v hlavním oknu. Dále je zde ukázáno, jak definovat funkce, podmínky a přiřazování obrázku do *Labelu*.

Kód

Po importování *tkinteru* a vytvoření hlavního okna do proměnné *root* můžeme začít pracovat na samotné úloze. Abychom vykreslili samotné okno i všechny widgety, musíme na *root* zavolat metodu *mainloop*. Samotná *mainloop* je nekonečná smyčka událostí, kterou vykreslujeme hlavní okno. Zapisujeme ji na poslední řádek kódu. Jako první věc je důležité vytvořit jednotlivé widgety. Pro rozdělení na dvě části je řešeno přes widgety *Frame*. Nastavujeme jim určité parametry. Jako první a povinný argument je rodič jiného widgetu. Jednoduše řečeno určujeme, jestli se prvek bude vkládat do jiného. Tento parametr se zapisuje na první pozici a jeho hodnota je název rodičovského widgetu. U *Framů* je to *root*. Dalším parametrem je definována barva pozadí pomocí *bg*, do které přiřazujeme modrou barvu ("*blue*"). Dále jsou zde vytvořeny *Labely*, které mají nastavený parametr svého rodiče (*frame1* nebo *frame2*). Dále nastavujeme parametr *text*, do které vkládáme daný text, například *PGM1P*. Samotnému textu pomocí parametru *font* nastavujeme vzhled písma, konkrétně styl, velikost písma, podtržení slova a tučné písmo. Poslední parametry jsou *fg* a *bg*, první z nich je nastavení barvy písma ("*pink*") a druhý pro pozadí samotného widgetu. U druhého *Labelu* nastavujeme stejné parametry s rozdílnými hodnotami. Přesouváme se k tlačítkům. Všechny tlačítka mají uvedeného rodiče. Jelikož je chceme umístit do prvního *Framu*, napíšeme *frame1*. Všechny tyto *Buttony* mají parametr *text*, do kterého uvádíme

název tlačítek ("*Obrázek1*", "*Obrázek2*", "*Změň*"). Posledním parametrem je *command,* díky kterému po stisknutí tlačítka se spustí daná funkce. Nyní jsme vytvořili jednotlivé widgety, ale musíme je umístit do hlavního okna. Je více možností umístění jednotlivých widgetů do hlavního okna. Pro nastavení pozice widgetů vně *Framů* je využíváno metody *grid*. Tato metoda se skládá z několika parametrů, s nimiž uspořádáme jednotlivé widgety do maticového tvaru. Využíváme zde řádku (*row*) a sloupce (*column*). Jim dále přiřazujeme hodnoty, podle kterých se jednotlivé widgety umisťují. Například u *label_nadpis1* určujeme pozici nultý řádek a třetí sloupec. U některých widgetů je možné se setkat i s parametry *padx* a *pady*. Tyto parametry se využívají pro odsazení šířky nebo výšky od ostatních widgetů dle zadaných hodnot. Po umístění jednotlivých widgetů je nutné vykreslit jejich rodiče neboli *Framy*. K tomu využijeme metodu *pack*.

```
frame1=tk.Frame(root,bg="blue")
frame2=tk.Frame(root,bg="blue")
label_nadpis1 = tk.Label(frame1,text="PGM1P",font= ('Arial 18 underline bold') ,fg="pink",bg="blue")
label_nadpis2=tk.Label(frame1,text="Jan Plechatý",font=("Arial 14 bold"),fg="white",bg="blue")
button_Obrazek1=tk.Button(frame1,text="Obrázek1",command=obrazek1)
button_Obrazek2=tk.Button(frame1,text="Obrázek2",command=obrazek2)
button_zmena=tk.Button(frame1,text="Změň",command=zmena)
label_Obrazky = tk.Label(frame2,image=obrazek_prvni,height=300,width=450)

label_nadpis1.grid(row=0,column=3)
label_nadpis2.grid(row=2,column=3)
button_Obrazek1.grid(row=3,column=2,padx=10)
button_Obrazek2.grid(row=3,column=3,padx=10)
button_zmena.grid(row=3,column=4,padx=10)
label_Obrazky.grid(row=10,column=3,pady=20)
frame1.pack()
frame2.pack()
root.mainloop()
```

Obrázek 19 Ukázka nastavení widgetů (Zdroj: vlastní)

Nyní máme vytvořené všechny widgety. Před funkcemi deklarujeme proměnné, které využijeme pro uložení obrázku. Do proměnné *obrazek_prvni* přiřazujeme *obrazek.png*. Pro přiřazení obrázku je využíváno třídy *PhotoImage*, ve které je parametr *file*, jenž se využívá pro specifikování cesty k obrázku. Stejný postup provedeme i pro druhý obrázek. Nyní se dostáváme k funkci *obrazek1*. Deklarujeme jednotlivé funkce klíčovým slovem *def*, názvu, kulatých závorek a dvojtečky. Název funkce je *obrazek1*, ve které na *label_Obrazky* voláme metodu *config*, kterou jsme schopni změnit obrázek *label_Obrazky*. Tuto úpravu provádíme v kulatých závorkách metody. Konkrétně zde nastavujeme parametr *image* na obrázek uložený v proměnné. Tento postup provedeme v obou zmíněných funkcích, s rozdílem přiřazovaného obrázku. V poslední funkci s názvem *zmena* pracujeme s funkcí

print, díky které si můžeme vypsat různé hodnoty do terminálu. V této ukázce konkrétně získáváme hodnotu obrázku z widgetu *label_Obrazky*. K této akci je potřeba využít metodu *cget*. Dále uvádíme podmínku, kdy se dotazujeme, zda hodnota *image* příslušného *Labelu* se rovná *pyimage2*. Na hodnotu *pyimage2* se dotazujeme, protože vnitřní názvy obrázků jsou *pyimage1* a *pyimage2*. Názvy jsme si zjistili přes funkci *print*. Podmínky deklarujeme pomocí klíčového slova *if* a ptáme se, zda hodnota *Labelu* se rovná danému obrázku, následuje dvojtečka. Pokud se podmínka splní (*True*) změníme hodnotu parametru *image* v *label_Obrazky* na *obrazek_prvni*. Jestliže se nesplní (*False*) provede se kód v bloku *else*. Jinak řečeno změníme obrázek *Labelu* na *obrazek_druhy*.

```
obrazek_prvni= tk.PhotoImage(file="obrazky_galerie/obrazek.png")
obrazek_druhy=tk.PhotoImage(file="obrazky_galerie/giphy.gif")

def obrazek1():
    label_Obrazky.config(image=obrazek_prvni)

def obrazek2():
    label_Obrazky.config(image=obrazek_druhy)

def zmena():
    print(label_Obrazky.cget("image"))
    if label_Obrazky.cget("image")=="pyimage2":
        label_Obrazky.config(image=obrazek_prvni)
    else:
        label_Obrazky.config(image=obrazek_druhy)
```

Obrázek 20 Ukázka přiřazení obrázků a funkcí (Zdroj: vlastní)

Obrázek 21 Výsledek Galerie Python (Zdroj: vlastní)

3.5.2 ÚLOHA 2

Popis úlohy

V úloze Smajlíci je ukázáno vytváření widgetů využívající se pro vstup od uživatele. Dále jsou ukázány jiné možnosti pozicování widgetů do hlavního okna. Pracuje se zde s proměnnými, parametry a vypisování hodnot do *Labelu*. V úloze jsou využívány různé typy dialogových oken. Jsou ukázány i různé práce s textem, například mazání posledního, prvního znaku nebo celého textu.

Kód

Importujeme *tkinter* a vytvoříme hlavní okno a přiřadíme ho proměnné *root*, kterému nastavíme rozměry a titulek. Dále je nutné spustit hlavní smyčku pomocí metody *mainloop*. Nyní si vytvoříme widgety přes jednotlivé třídy z modulu *tkinter* a přiřadíme je do proměnných. Jako první jsou zde tlačítka. Přiřazujeme jim rodiče (*root*) a nastavíme jim parametr *text* na daného smajlíka. U všech tlačítek je parametr *command*, kterému přiřazujeme danou funkci. Výjimka je u funkcí, které mají parametr, kde využíváme anonymní funkce *lambda*. Samotná *lambda* zavolá danou funkci, které přiřazuje hodnotu (argument) smajlíka. Jako další widget je zde *Entry*, který se používá pro získání hodnot od uživatele. Přiřazujeme mu pouze rodiče. Po vytvoření widgetů je důležité provést jejich umístění. Je zde využito metody *pack* a *place*. Oběma metodám jsou přiřazené parametry s hodnotami. Jako první si rozebereme *pack*. Využíváme parametrů *side* a *anchor*. Díky *side* můžeme widget zarovnat na určitou stranu. V této úloze je přiřazená hodnota *"left"*, to znamená, že všechny widgety, které mají tuto hodnotu ve zmíněném parametru se budou zarovnávat vlevo podle pořadí vykreslení. *Anchor* neboli ukotvení využíváme pro vykreslování jednotlivých widgetů v levém horním rohu. Proto má přiřazenou hodnotu *"nw".* Používají se zde i jiné hodnoty a jsou psány ve zkratkách světových stran z angličtiny (*nw = north west)*. V metodě *place* uvádíme parametry *x* a *y*, do kterých přiřazujeme konkrétní pozice vykreslení.

```
smejici_button=tk.Button(root, text=":-D", command=smejici)
slzici_button=tk.Button(root, text=":'-)", command= lambda:univerzalni(":'-)"))
```

Obrázek 22 Ukázka lambda (Zdroj: vlastní)

```
usmevavy_button.pack(side="left",anchor="nw")
```

Obrázek 23 Ukázka side a anchor (Zdroj: vlastní)

```
label_smajl.place(x=10,y=50)
root.mainloop()
```

Obrázek 24 Ukázka place a mainloop (Zdroj: vlastní)

Nyní se dostáváme k popisu jednotlivých funkcí. Deklarujeme je pomocí klíčového slova *def*, názvu funkce a kulatých závorek. První funkce má název *usmevavy*. Je zde deklarovaná proměnná *usmev*, do které je přiřazen smajlík. Hodnotu parametru *text label_vypis* získáme pomocí metody *cget* a přiřazujeme ji do proměnné *fronta*. Pro výpis smajlíka a *fronty* do *label_vypis* je volána metoda *config*. V této metodě do parametru *text* přiřazujeme dvě vytvořené proměnné a pro jejich zřetězení využíváme operátoru plus. Pro vypsání smajlíka do *label_smajl* je také využita metoda *config* a parametr *text*. Následuje funkce s názvem *smutny*. V ní deklarujeme proměnnou *smutek*, do které přiřazujeme hodnotu parametru *text* tlačítka *smutny_button*. Následující kód funkce je stejný jako u předešlé. Další funkce se nazývá *smejici*, ve které je deklarovaná proměnná *fronta*. Rozdíl oproti předešlým funkcím je přiřazení statické hodnoty smajlíka do metody *config*. Funkce *univerzalni* obsahuje parametr, který nabývá smajlíka podle toho, jaké tlačítko je stisknuto. Se samotným parametrem je v kódu pracováno jako s proměnnou.

```
def usmevavy():
    usmev = "😂"
    fronta = label_vypis.cget("text")
    label_vypis.config(text=fronta+" "+usmev)
    label_smajl.config(text=usmev)
```

Obrázek 25 Ukázka funkce usmevavy (Zdroj: vlastní)

```
def univerzalni(smail):
    fronta = label_vypis.cget("text")
    label_vypis.config(text=fronta+" " + smail)
    label_smajl.config(text=smail)
```

Obrázek 26 Ukázka funkce univerzalni s parametrem (Zdroj: vlastní)

Přesouváme se k funkcím, kterými „historii“ smajlíků upravujeme mazáním nebo přidáváním textu od uživatele. První je zde funkce s názvem *mazani*, ve které přiřadíme textovou hodnotu *label_vypis* pomocí metody *cget* do proměnné *fronta*.

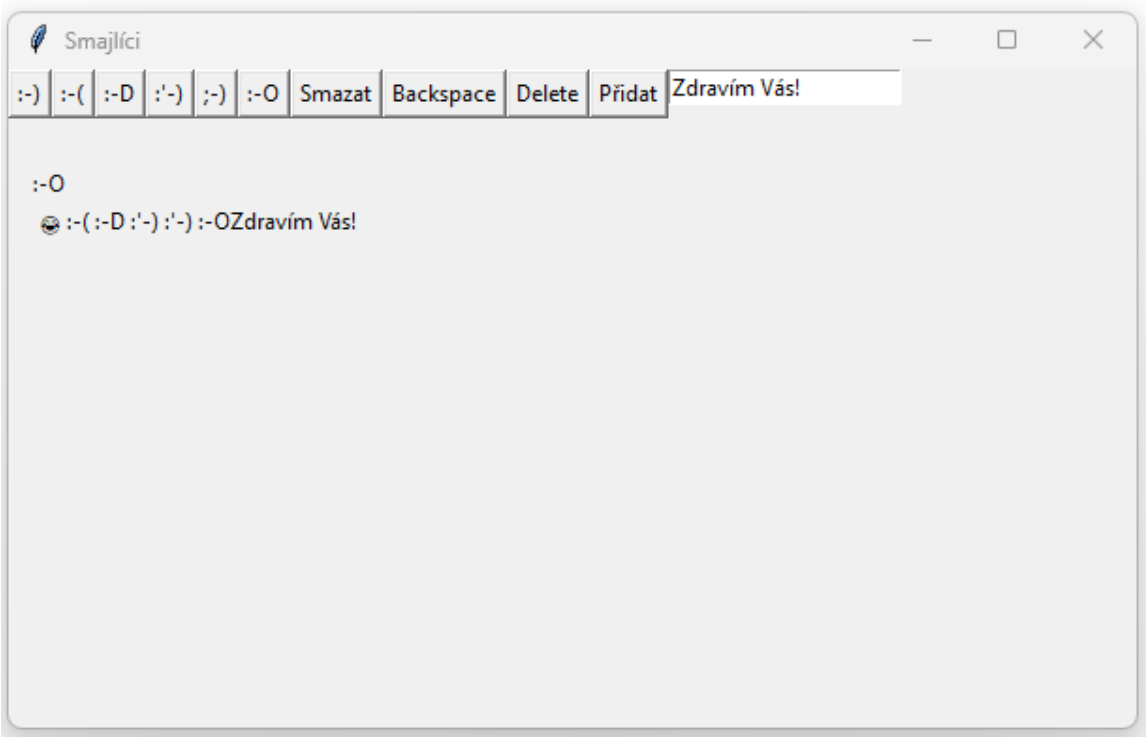
Uvádíme podmínku, ve které se ptáme, zda proměnná *fronta* nabývá prázdné textové hodnoty. Pokud je to pravda (*True*), tak se zobrazí dialogové okno (*messagebox*). Pokud jsme neimportovali z *tkinteru* vše hvězdičkou, musíme ho ručně importovat (*from tkinter import messagebox*). *Messageboxu* můžeme přiřazovat různé funkce, díky kterým můžeme rozlišit, jestli se jedná o error, varování nebo info. V této podmínce je využita funkce *showerror*. Využíváme zde dvou parametrů, kdy do prvního uvádíme titulek dialogového okna a druhému sdělení uživateli. Pokud se podmínka vyhodnotila jako nepravda, tak oběma parametrům *text* zmíněných *Labelů* přiřadíme hodnotu prázdného textu. Funkce *backspace* má za úkol mazat poslední znak řetězce v *label_vypis*. Jako v předchozích funkcí je zde deklarovaná proměnná *fronta* a stejná podmínka. *Messagebox* se spustí, pokud vyhodnocení *if* je pravda, jinak se deklaruje nová proměnná s názvem *vypsani*. Do této proměnné ukládáme hodnotu řetězce *fronta[:-1]*. To znamená, že se přiřadí všechny hodnoty řetězce kromě posledního znaku. Téměř stejně je i napsaný kód pro funkci *delete* s rozdílem, že do proměnné *vypsani* přiřazujeme hodnotu *fronta[1:]*. Tento zápis znamená, že se vynechá první znak řetězce. Jako poslední funkci zde máme *pridat*. Deklarujeme proměnnou *vstup*, do které přiřazujeme hodnotu vstupu od uživatele (*vstup_entry*) pomocí metody *get*. V následující podmínce se ptáme, zda vstup není prázdný. Pokud je tato podmínka splněna, přes *messagebox* uživateli sdělujeme text, který bude vypsán do *label_vypis*. Dále deklarujeme proměnnou *fronta*, do které přiřazujeme hodnotu získanou přes parametr *text*. Obě tyto proměnné jsou dále přiřazeny do parametru *text label_vypis*. Jestliže podmínka bude vyhodnocena jako nepravda, uživatel bude upozorněn *messageboxem*.

```
def backspace():
    fronta = label_vypis.cget("text")
    print(fronta)
    if fronta == "":
        messagebox.showerror("Upozornění","Není co mazat!")
    else:
      #length=len(fronta)
      #vypsani=fronta[:length-1]
      vypsani = fronta[:-1]
      label_vypis.config(text=vypsani)
```

Obrázek 27 Ukázka funkce backspace (Zdroj: vlastní)

```
def pridat():
    vstup =vstup_entry.get()
    if vstup != "":
        messagebox.showinfo("Info","Přidáváte tento text: " + vstup)
        fronta = label_vypis.cget("text")
        label_vypis.config(text= fronta+vstup)
    else:
        messagebox.showwarning("Varování", "Není co vložit!")
```

Obrázek 28 Ukázka funkce pridat (Zdroj: vlastní)



Obrázek 29 Výsledek Smajlíci Python (Zdroj: vlastní)

3.5.3 ÚLOHA 3

Popis úlohy

Nejprve je nastavení vzhledové části úlohy, kdy si jednotlivé widgety vytvoříme a nastavíme jim parametry. Poté je umístíme do hlavního okna. V logické části je ukázána práce s hodnotami textového typu. Jsou zde funkce, kterými text různými způsoby mažeme, upravujeme a nahrazujeme. S tím souvisí i práce s globální proměnnou a *listem*. Dále pracujeme s regulárními výrazy, různými moduly pythonu a podmínkami.

Kód

Provedeme importování *tkinteru* a *messageboxu*. Pro *tkinter* nastavíme *title* a *geometry*. Nezapomeňme spustit smyčku *mainloop*. Samotnou úlohu rozdělíme na dvě části pomocí widgetů *Frame* a uložíme si je do proměnných. Do parametrů jim nastavíme rodiče *root*, výšku a šířku. Dále vytvoříme *Label*, který se bude vykreslovat v *ramecek1* a nastavíme mu parametr *text*. Jelikož se jedná o „nadpis“ první části, přiřadíme do parametru *font* styl, velikost a tučné písmo. Do *ramecek1* chceme vytvořit widget, do kterého bude možné zapisovat text. Tento widget se nazývá *Text* a definujeme mu rodiče *ramecek1*, výšku a šířku. Abychom na něj mohli odkazovat je důležité widgety ukládat do proměnných. Oba tyto prvky jsou vykresleny do prvního *Framu* pomocí metody *pack*. U *label1* je nastaveno *pady*, aby widgety nebyly blízko sebe. Nyní si popíšeme widgety, které se budou vykreslovat v *ramecek2*. Jako první je *Label*, který využijeme pro zobrazení nadpisu. Je potomkem *ramecek2*, má nastavený parametr *text* a *font*, který má stejné hodnoty jako *label1*. Pod „nadpisem“ jsou *Buttony*, které mají nastaveného rodiče *ramecek2*, parametr *text* s určitým textem a *command*. Ve druhém *Framu* se využívá i widget *Text*. Do něj převezmeme text od uživatele, jenž budeme schopni různě upravovat. Widgety *Label* a *Buttony* se vykreslují metodou *grid*, do které přiřadíme hodnoty řádce a sloupečku. U tlačítek je parametr *padx* a *pady*, kterými odsazujeme widget od ostatních o zadané hodnoty. Zmíněný *Text* umisťujeme pomocí metody *pack*.

Nyní si popíšeme jednotlivé funkce. Jako první si představíme funkci *prevzit_text*. Touto funkcí budeme přebírat hodnotu z widgetu *textarea1* do *textarea2*. Deklarujeme v ní proměnnou *text* a přiřazujeme jí hodnoty, nacházející se v *textarea1*. Hodnoty získáme pomocí metody *get*, které v první části nastavujeme začátek a druhé konec. Ve druhé části si můžeme všimnout *end-1c*. Pomocí tohoto zápisu se nám do celkové délky nebude započítávat odřádkování (*„/n“*). Následuje větvená podmínka, ve které se ptáme, jestli v *text* je alespoň jeden znak. K tomu je využita funkce *len*, jenž nám vrací délku řetězce zadanou v jejím parametru. Pokud je to pravda, je zde další podmínka, ve které se ptáme, jestli *text* má méně než sto znaků. Jestliže je vyhodnocena jako *True*, tak se zobrazí *messagebox*, vněm je řetězec, který vypíše oznámení a přesnou délku textu. Vkládání číselné hodnoty je nutné v *messageboxu* přetypovat na *str*.

```
Jeho přesná délka je " + str(len(text)) + ".")
```

Obrázek 30 Ukázka přetypování délky proměnné text (Zdroj: vlastní)

Pokud by se podmínka nevyhodnotila jako pravda, následuje další větev (*elif*), ve které se ptáme, jestli je text delší nebo rovno 100 a zároveň menší nebo rovno 200 znaků. Pokud je to pravda, tak se zobrazí *messagebox* s daným textem a přesnou délkou textu. Jinak se zobrazí *messagebox* s příslušným textem a přesným počtem znaků. Pokud uživatel do *textarea1* neuvedl žádný znak, tak se zobrazí *messagebox*, sdělující uživateli, aby napsal nějaký text. Do teď byla podmínkami kontrolována délka textu a její případný komentář. Zatím jsme do *textarea2* nepřevedli žádný text. Proto na proměnnou *textarea2* je volána metoda *insert*, ve které v prvním parametru uvádíme začátek vypisování do widgetu. V druhém parametru uvádíme proměnnou *text*. Další funkce, které si budeme popisovat mají za úkol nahradit různé znaky podtržítkem. V každé funkci je využit jiný způsob. Jako první z nich je zde funkce *y_i_funkce*. V této funkci je nahrazování prováděno statickým způsobem. Nahrazujeme zde všechny zápisy *i* a *y* nacházející se v *textarea2* za podtržítka. Pro všechny písmena je vytvořena proměnná, do které přiřazujeme *textarea2* s metodou *search*. V první části metody je uveden znak, který hledáme a do druhého kde se má začít s hledáním. Pokud se daný znak vyhledá, do proměnné se přiřadí pozice nalezeného znaku, jinak prázdný řetězec. Následuje podmínka, ve které mezi jednotlivé proměnné použijeme operátor *or*. Jinými slovy, pokud se ze všech hledaných znaků alespoň jeden našel, tak se provede blok kódu. V tomto bloku je pro uvedené znaky na widgetu *textarea2* volána metoda *replace*. V prvním parametru udáváme, na jaké pozici se znaky mají začít nahrazovat a ve druhém kde skončit. Do posledního parametru se uvádí *str*, kterým chceme nahradit část, v našem případě všechny hodnoty ve widgetu *textarea2*. Proto zde využijeme proměnnou *textarea2*, na které se volá metoda *get*. Tím získáme vše, co se v textarea2 nachází a zároveň využijeme metody *replace*. Do této metody jako první parametr uvádíme text, který chceme nahradit a do druhého čím se má nahradit. Pokud se v textu žádné *i* nebo *y* nevyskytuje, tak se zobrazí uživateli *messagebox*.

```
textarea2.replace("1.0", "end",textarea2.get("1.0", "end").replace("y", "_"))
textarea2.replace("1.0", "end",textarea2.get("1.0", "end").replace("i", "_"))
```

Obrázek 31 Ukázka nahrazování textu (Zdroj: vlastní)

Další funkce s názvem *s_z_funkce* nahrazuje všechny výskyty *s* a *z* ve widgetu *textarea2* za podtržítka. Deklarujeme zde proměnnou *text* a přiřazujeme ji obsah z *textarea2*. A metodou *lower* nastavíme všechny znaky na malé. Následně všechny hodnoty ve widgetu smažeme a poté mu přes metodu *insert* nastavíme hodnotu *text*. To znamená, že se vypíše text malými písmeny. Poté si deklarujeme proměnnou *textS* a *textZ*. Do nich přiřazujeme *textarea2.search*, kdy v prvním parametru je uveden hledaný znak a druhému pozice, od které se má začít hledat. Do proměnných *textS* a *textZ* jsou přiřazeny pozice nalezených znaků nebo prázdný řetězec na základě výskytu. Dále je uvedená podmínka, ve které se ptáme, jestli se našli oba znaky. Pokud ano jsou nahrazeny, jinak se dotazujeme (*elif*), zda se našel alespoň znak *s*. Jestliže tomu tak je, jsou nahrazeny všechny tyto znaky za podtržítka. Pomocí *elif* se stejně ptáme i na znaky *z* a pokud ani ten není nalezen, je zobrazen *messagebox*. Nyní se dostáváme k nejvhodnějšímu a nejpokročilejšímu nahrazování znaků. Ve funkci *me_mne_funkce* využíváme regulárních výrazů. Nejprve si deklarujeme proměnnou *text* a uložíme do ní hodnotu z *textarea2*. Následuje podmínka, ve které definujeme, jestli se v *text* nachází *mě* nebo *mně*. Abychom mohli s regulárními výrazy pracovat, musíme importovat modul *re*. V podmínce na modul *re* voláme metodu *search*. V ní uvádíme regulární výrazy (*mě* a *mně*). V druhém parametru určujeme oblast, kde se výrazy budou hledat. Samotné výrazy se zapisují do závorek a mezi nimi je znak /, který se využívá pro operaci nebo. Abychom vyhledávali regulární výrazy napsané malými a velkými písmeny, je zde využit zápis (*?i*). Tato zkratka se používá pro „ignore case“ neboli ignorování velkých a malých písmen u regulárních výrazu. Pokud se alespoň jeden ze znaků v *textu* nachází, je deklarovaná proměnná *replaced_text*, do které uložíme upravený text. Toho docílíme pomocí modulu *re* a funkce *sub*. Do této funkce v prvním parametru uvádíme regulární výrazy, v dalším je znak ("_"), kterým chceme znaky nahradit. Ve třetím vkládáme oblast nahrazování. U regulárních výrazů je znovu využit zápis (*?i*).

```
if(re.search(("(?i)(mě|mně)"),text)):
    replaced_text = re.sub("(?i)(mě|mně)", "_", text)
```

Obrázek 32 Ukázka regulárních výrazů (Zdroj: vlastní)

Poté hodnoty v *textarea2* smažeme a vložíme do ní proměnnou *replaced_text*. Po nahrazení znaků podtržítky, chceme přes *Button* vrátit původní hodnotu. Abychom uživateli zdůraznili znaky, které byly nahrazeny podtržítkem, vrátíme jim text

v upravené podobě, konkrétně velkými písmeny a zvýrazněny červenou barvou. Tato funkce má název *puvodni*, ve které deklarujeme proměnnou *area_text*. Do ní přiřazujeme hodnoty z *textarea2*. Musíme ve druhém parametru uvést *"end-1c"*, protože jsme ji uvedli i v *textarea1*. Než začneme s nahrazováním podtržítek, musíme se v podmínce zeptat, jestli vůbec k nějakým úpravám došlo. Pokud bychom v podmínce využili hodnot z *textarea1*, nemusela by podmínka být přesná. Je to z důvodu toho, že uživatel by do prvního widgetu *Text* mohl dopsat nějaký text a tím by podmínku splnil. Abychom této situaci předešli, je důležité deklarovat globální proměnnou *puvodni_text*. V kódu je deklarována nad všemi funkcemi. Do této proměnné ve funkci *prevzit* přiřadíme hodnotu *textarea1*. Na globální proměnnou je důležité ve funkci odkazovat klíčovým slovem *global* a jejího názvu a až na dalších řádcích přiřazovat hodnoty. Jakmile jsme převzali text je možné porovnávat hodnotu mezi *area_text* a *puvodni_text*. Jestliže nejsou stejné, tak smažeme hodnotu v *textarea2*. Deklarujeme proměnnou *znaky_uprava*, které přiřadíme příslušné regulární výrazy. Dále deklarujeme proměnnou *new_text* a do ní přiřazujeme text, ve kterém se výrazy převedou na velká písmena. Toho docílíme pomocí modulu *re* a funkcí *sub*. Touto funkcí je možné nahradit všechny výskyty regulárních výrazů. Do prvního parametru vkládáme proměnnou *znaky_uprava*. V druhém parametru je uvedena anonymní funkce *lambda* s *m*, které představuje match objekt. Díky nim dokážeme lokalizovat znak, který chceme nahradit. Jako třetí parametr je uveden *puvodni_text*, v němž jednotlivé výrazy hledáme. Funguje to tak, že funkce *sub* vyhledá první z výrazů v proměnné *puvodni_text*, který se má nahradit. Poté se zavolá anonymní funkce a do *m* se přiřadí pozice výskytu znaku. Dále je zavolána metoda *group*, která vrátí výraz v textové podobě a pomocí metody *upper* mu nastavíme velké písmeno. Tyto změny jsou následně uloženy do proměnné *new_text*. Tato funkce se opakuje pro každý znak. Hodnota v *puvodni_text* se nijak nezměnila.

```
new_text = re.sub(znaky_uprava, lambda m: m.group().upper(), puvodni_text)
```

Obrázek 33 Ukázka funkce sub (Zdroj: vlastní)

Následně vypíšeme text z proměnné *new_text* do widgetu *textarea2*. Ještě nám zbývá jednotlivým znakům nastavit červenou barvu. Proto je zde vytvořena funkce *podbarveni* s parametrem *text*, kterou voláme ve funkci s názvem *puvodni*. Do jejího parametru je přiřazena proměnná *new_text*. V *podbarveni* jako první vytvoříme tag na widgetu

textarea2, kterému přiřadíme název *barva* a vlastnost *foreground="red"* neboli červenou barvu textu. Zatím jsme ji ale žádnému textu nepřiřadili. Dále si deklarujeme proměnnou *vyskyty*. Hledání výrazů je uskutečněno přes funkci *finditer*. Do prvního parametru vkládáme znaky, které chceme najít a do druhého *text*, který nám určuje řetězec pro hledání. Funkce *finditer* vrátí match objekty, ve kterých jsou uloženy pozice nalezených regulárních výrazů.

```
vyskyty = re.finditer("(?i)(s|z|i|y|ý|í|mě|mně)",text)
```

Obrázek 34 Ukázka finditer (Zdroj: vlastní)

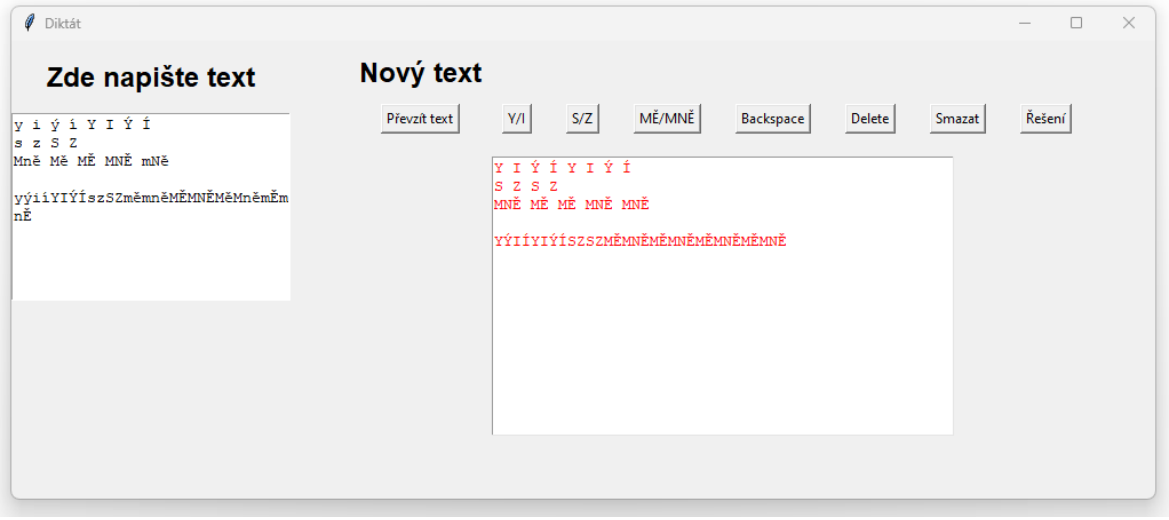
Nyní musíme vyřešit, jak všem nalezeným regulárním výrazům ve *vyskyty* přiřadit barvu. Pro přiřazení využijeme *for* cyklu, ve kterém projdeme všechny objekty obsahující nalezené výskyty. Z objektu *vyskyt* naplníme proměnné *start_index* a *konec_index* příslušnými hodnotami, které zformátujeme na řetězec pomocí malého *f*. Díky tomu můžeme vkládat počáteční (*vyskyt.start()*) a konečné (*vyskyt.end()*) hodnoty pozice znaku do řetězce. V zápisu řetězce se využívá počáteční hodnoty widgetu (*1.0*) a malého *c*. Jako poslední věc je zde obarvení textu pomocí metody *tag_add* volanou na widgetu *textarea2*. Na první pozici v něm určujeme, jaký tag chceme přidat a do ostatních vkládáme počáteční a poslední pozici znaku, jenž chceme obarvit.

```
start_index = f"1.0+{vyskyt.start()}c"
konec_index = f"1.0+{vyskyt.end()}c"
textarea2.tag_add("barva", start_index, konec_index)
```

Obrázek 35 Obarvení textu pomocí tag (Zdroj: vlastní)

V úloze jsou dále funkce, ve kterých mažeme text různými způsoby. Jako první je zde funkce *mazani*. Deklarujeme proměnnou *fronta*, do které přiřadíme hodnoty z widgetu *textarea2*. V podmínce se ptáme, jestli je *fronta* rovna prázdnému řetězci, pokud ano, zobrazí se *messagebox* s textem *„Není co mazat!“*. Jinak se *textarea2* smaže. Dále je zde funkce, ve které mažeme poslední znak s názvem *backspace*. Do proměnné *fronta* přiřazujeme hodnotu widgetu a deklarujeme zde stejnou podmínku. Pokud widget není prázdný, smažou se hodnoty v *textarea2*. Dále se deklaruje proměnná *vypsani* do které se ukládají hodnoty řetězce z proměnné *fronta* o jednu menší (poslední znak je vynechán). Přes metodu *insert* volanou na *textarea2* je vložena proměnná *vypsani*. Poslední funkce této úlohy je *delete*. Tato funkce má kód téměř stejný jako *backspace*. Rozdíl zde nastává

v přiřazování hodnoty do proměnné *vypis*, kdy začínáme vkládat řetězec v proměnné *fronta* od druhého znaku.



Obrázek 36 Výsledek Diktát Python (Zdroj: vlastní)

3.5.4 ÚLOHA 4

Popis úlohy

V úloze Ovocné hrátky si ukážeme vytvoření jednotlivých widgetů a využití jednotlivých parametrů pro pozicování a odsazení od dalších prvků. Dále je ukázána práce s *listem*, kde si ukážeme jejich využití pro ukládání hodnot. V poslední části se dotazujeme, zda widget obsahuje číselnou, kde je využíváno přetypování na datový typ *int*.

Kód

Importujeme modul *tkinter*, z kterého vytvoříme hlavní okno a nastavíme titulek a *geometry*. Z něho importujeme modul *messagebox*. Nyní si vytvoříme jednotlivé widgety. Jako první vstup pro uživatele neboli *Entry*, uložíme si ji do proměnné *entry_cisloOvoce* a přiřadíme rodiče *root*. Dále zde máme tlačítko s názvem *odeslat_button* s rodičem *root* a parametry *text* a command. Ten má přiřazenou funkci *zobrazit*. Poslední widget, který je zde využíván je *Label*. Tento widget má pouze parametr rodiče. Samozřejmě je zde využito více vstupů, tlačítek a *Labelů* s rozdílnými v parametry *text* a *command*.

```
entry_cisloOvoce=tk.Entry(root)
odeslat_button=tk.Button(root,text="Odeslat",command=zobrazit)
ovoce_label=tk.Label(root)
```

Obrázek 37 Ukázka widgetů (Zdroj: vlastní)

Nyní se dostáváme k pozicování jednotlivých widgetů. Řešíme to zde metodou *grid* a u *vypsaniPole_label* metodou *place*. Samotný *grid* využíváme pro uspořádání do řádků a sloupečků. Do této metody využíváme parametry *column*, *row*, *padx* a *pady*. Do parametru *column* přiřazujeme číslo, které nám reprezentuje sloupec. S tím souvisí parametr *row*, který nám určuje řádek. Abychom neměli jednotlivé widgety blízko sebe, přiřazujeme některým parametry *padx* a *pady*. Pro umístění již zmíněného *Labelu* používáme metodu *place*, do které uvádíme parametry *x* a *y*.

Dostáváme k logické části úlohy. Jako první je důležité nadefinovat *list* s názvem *ovoce*, ve kterém jsou hodnoty (názvy) různých druhů ovoce textového typu. Následuje deklarování jednotlivých funkcí. Ve funkci *zobrazit* se dotazujeme, zda do daného *Entry* uživatel napsal číslo. Pro tuto podmínku je využito proměnné, která odkazuje na vstup a voláme na ni dvě metody. První z nich je metoda *get*, pomocí které jsme schopni získat hodnotu v daném *Entry*. Následuje volání další metody tentokrát *isnumeric*, tou se zeptáme, jestli hodnota je číslo. Jestliže vyhodnocení podmínky je pravda, deklarujeme proměnnou *cisloOvoce*. Do té přiřazujeme hodnotu, která je ve *entry_cisloOvoce* pomocí metody *get*. S danou hodnotou budeme provádět aritmetickou operaci, a proto je nutné přetypovat hodnotu na číslo pomocí *int*. V dalším kroku měníme hodnotu *text Labelu* s názvem *ovoce_label* metodou *config*. Do tohoto parametru vkládáme název ovoce na konkrétní pozici v *listu*. Využíváme názvu proměnné, která odkazuje na *list* a do hranatých závorek vkládáme proměnnou *cisloOvoce* (pozici ovoce, jenž chce uživatel z *listu* vypsat). Od ní odečítáme číslo 1, protože pro uživatele první pozice je 1, ale pro *list* je to 0. Jelikož tuto hodnotu přiřazujeme do parametru *text*, je nutné přiřazující hodnotu přetypovat do textového typu (*str*). To byl popis kódu, který se provede, pokud podmínka je pravda. Jestliže se tato podmínka vyhodnotí jako nepravda, je *messagebox* s funkcí *showwarning*, obsahující text, který se uživateli zobrazí.

```
def zobrazit():
    if (entry_cisloOvoce.get().isnumeric()):
        cisloOvoce = int(entry_cisloOvoce.get())

        ovoce_label.config(text=str(ovoce [cisloOvoce-1]))
```

Obrázek 38 Ukázka funkce zobrazit (Zdroj: vlastní)

Než si popíšeme funkce *vlozitKonec* a *vlozitZacatek*, tak si rozebereme kód funkce *smazatvolitelne*, protože zápisem si jsou podobné se *zobrazit*. Pomocí této funkce chceme mazat ovoce na pozici, kterou nám zadá uživatel. Je zde využita stejná podmínka, kdy při kladném vyhodnocení na proměnnou ovoce voláme metodu *pop*, do které je nutné zadat číselnou hodnotu. Z toho důvodu hodnotu *entry_smazatPozici* získanou metodou *get* je nutné přetypovat na *int*. Při nesplnění podmínky se zobrazí *messagebox* s funkcí showwarning.

```
def smazatvolitelne():
    if (entry_smazatPozici.get().isnumeric()):
        ovoce.pop(int(entry_smazatPozici.get())-1)
    else:
        messagebox.showwarning("Upozornění", "Vložte pozici ovoce!")
```

Obrázek 39 Ukázka funkce smazatvolitelne (Zdroj: vlastní)

Nyní se přesouváme na vynechané funkce, jako první si popíšeme *vlozitKonec*. Zadaný text uživatelem chceme vložit na konec *listu*. Je zde podmínka, ve které se ptáme, jestli je hodnota z *Entry* číslo. Pokud je to pravda, zobrazí se upozornění s textem „*Vložte název ovoce!*“. Jestliže v *entry_pridejte* není číslo, tak na proměnnou *ovoce* je volána metoda *append*, do které přiřazujeme hodnotu *entry_pridejte* metodou *get*. Nyní si popíšeme funkci *vlozitZacatek*. Podmínka je stejná jako u předešlé funkce. Rozdíl zde nastává v přidávání textu na začátek *listu*. Na proměnné *ovoce* využijeme metodu *insert*, do které jako první argument zadáváme pozici. Je zde uvedena nula, protože *list* začíná od nuly. Do druhého argumentu vkládáme hodnotu zadanou uživatelem. Dále jsou zde funkce, kterými mažeme poslední nebo první hodnotu v *listu*. Ve funkci *smazatposledni* je na *list ovoce* volána metoda *pop*. Pokud do *pop* neuvedeme parametr, automaticky maže poslední prvek. Ve funkci *smazatprvni* je tato metoda také volána s rozdílem, že zde uvádíme argument a tím je nula neboli určení pozice mazané hodnoty.

```
def vlozitKonec():
    if (entry_pridejte.get().isnumeric()):
        messagebox.showwarning("Upozornění", "Vložte název ovoce!")
    else:
        ovoce.append(entry_pridejte.get())
```

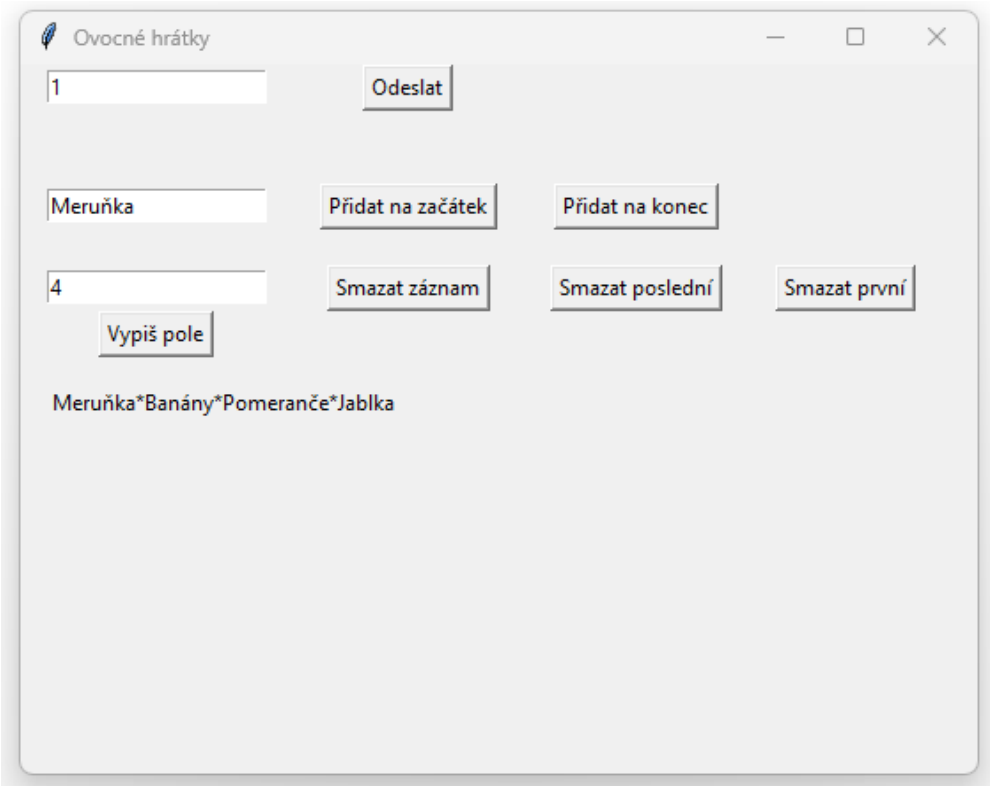
Obrázek 40 Ukázka funkce vlozitKonec (Zdroj: vlastní)

Poslední funkce s názvem *vypisPole* vypisuje do *vypsaniPole_label* všechny prvky z *listu*. Mezi jednotlivými prvky je hvězdička. Deklarujeme proměnnou *hvezdicka* a přiřazujeme jí

znak hvězdičky ("*"). Vytvoříme proměnnou *vypis*, do které jsou přiřazeny prvky z *listu*, mezi kterými je hvězdička. K tomu je využito metody *join*, díky které je možné vložit zmíněný znak. Dále následuje samotné přiřazení *vypis* do *vypsaniPole_label* metodou *config* a jejím parametrem *text*.

```
def vypisPole():
    hvezdicka="*"
    vypis=hvezdicka.join(ovoce)
    vypsaniPole_label.config(text=vypis)
```

Obrázek 41 Ukázka funkce vypisPole (Zdroj: vlastní)



Obrázek 42 Výsledek Ovocné hrátky Python (Zdroj: vlastní)

3.5.5 ÚLOHA 5

Popis úlohy

V úloze Kalkulačka je ukázáno vytváření widgetů a přiřazování hodnoty widgetu *Entry* při vykreslení. Dále přes funkce provádíme matematické operace, u kterých využíváme operátorů. V těchto funkcích je podmínka ověřující, zda jsou vstupy od uživatele čísla. V této úloze je funkce s parametry pro kontrolu čísel v jednotlivých *Entry*.

Kód

Importujeme *tkinter*, kdy nastavíme na proměnné *root title* a *geometry*. Dále z *tkinteru* importujeme *messagebox*. Nezapomeňme na hlavní smyčku *mainloop*. Nyní vytvoříme widgety. Jako první vytvoříme tlačítka. Přiřadíme je do proměnných s příslušnými názvy a nastavujeme jim parametry jako je *root*, *text* a *command*. Dále vytváříme *Entry*, ve kterých při spuštění hlavního okna v obou vstupech chceme zobrazovat text. Toho docílíme pomocí metody *insert*, kterou voláme na vytvořené proměnné *entry1* a *entry2*. Metodě *insert* přiřazujeme dva argumenty, do prvního určujeme počáteční pozici a druhému text, který chceme zobrazit. Poslední widget, který potřebujeme pro vypisování je *Label*, kterému nastavíme *root*. Jednotlivé widgety je nutné vykreslit do hlavního okna. Využíváme metody *grid*, ve které definujeme řádek (*row*) a sloupec (*column*).

```
entry1= tk.Entry(root)
entry1.insert(0,"číslo 1")
```

Obrázek 43 Ukázka metody insert (Zdroj: vlastní)

Nyní si rozebereme jednotlivé funkce. Jako první je zde funkce *scitani*. Deklarujeme proměnné *vstup1*, *vstup2* a přiřazujeme jim proměnné *entry1* a *entry2*. Dále musíme uvést podmínku, ve které se dotazujeme, zda hodnota v *entry1* a *entry2* je číslo. K získání hodnoty využijeme metodu *get* a poté voláme další metodu *isnumeric*. V podmínce si můžeme všimnout logického operátoru *and*. Podmínka se vyhodnotí jako *True*, pokud v obou vstupech bude číslo. Pokud je podmínka splněna deklarujeme proměnné *cislo1* a *cislo2* a přiřazujeme jim hodnoty *vstup1* a *vstup2*, na kterých je volána metoda *get*. Jelikož s těmito hodnotami chceme dále pracovat jako s čísly, je důležité je přetypovat. Může nastat situace, kdy uživatel bude chtít pracovat i s desetinnými čísly. Proto zde vstupy přetypováváme na datový typ *float*, který dokáže pracovat s plovoucí čárkou.

```
if(entry1.get().isnumeric()) and ((entry2.get().isnumeric())):
  cislo1 = float(vstup1.get())
  cislo2=float(vstup2.get())
```

Obrázek 44 Ukázka podmínky, operátoru and a float (Zdroj: vlastní)

Následuje deklarace proměnné *vysledek*, do které provádíme samotný výpočet mezi *cislo1* a *cislo2*. Pokud bychom nepřetypovali zmíněné proměnné, tak by hodnoty mezi sebou neprovedli výpočet, ale zřetězily by se do textu. Pro jednotlivé matematické operace se

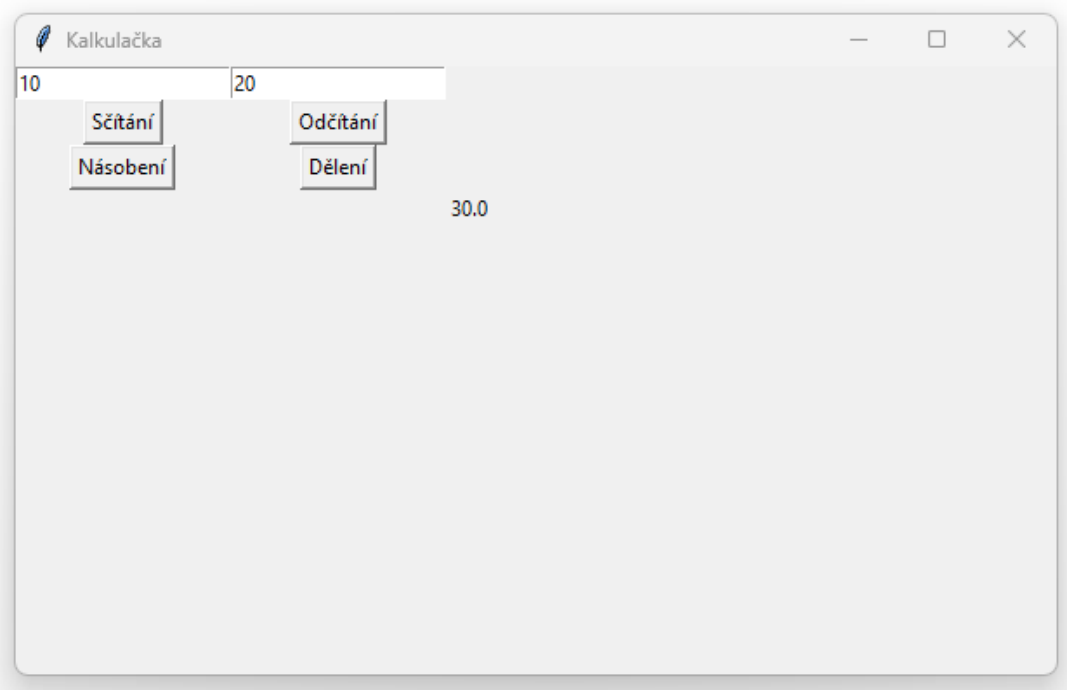
využívají operátory *+*, *-*, ***, */*. Po provedení výpočtu, do *label_vysledek* pomocí metody *config* parametru *text* přiřadíme proměnnou *vysledek*. Tento kód se provedl při vyhodnocení podmínky jako pravda. Pokud nastane situace, kdy uživatel nezadá do obou *Entry* číslo, tak je volána funkce *Nenicislo* s parametry *vstup1* a *vstup2*, do kterých přiřazujeme argumenty *vstup1.get()* a *vstup2.get()*, jinak řečeno hodnoty, které jsou v obou vstupech. Tato funkce má za úkol zjistit v jakém, případně obou vstupech uživatel nezadal číslo. Jelikož se potřebujeme zeptat, zda hodnoty nejsou číslo, je zde využito logického operátoru *not* neboli negace před podmínkou. V první podmínce se ptáme, jestli oba vstupy nejsou čísla, pokud je to pravda, zobrazí se *messagebox* s funkcí *showerror*. V ní je parametr titulek dialogového okna a text, který se má uživateli zobrazit. Pokud se podmínka vyhodnotí jako nepravda následuje *elif* neboli další podmínka, na kterou se ptáme, pokud výsledek předešlé podmínky nebyl *True*. Jestliže bychom využili pouhého *if*, tak by druhá podmínka proběhla bez ohledu na vyhodnocení první. Ve druhé podmínce se dotazujeme, zda první *Entry* není číslo. Jestliže ani tato podmínka se nevyhodnotí jako pravda následuje další *elif*, kde se ptáme, jestli druhý *Entry* není číslo. Jelikož funkce *Nenicislo* byla zavolána jednou z funkcí, je jisté, že jedna z podmínek bude pravda.

```
else:
    Nenicislo(vstup1.get(),vstup2.get())
```

Obrázek 45 Ukázka volání funkce Nenicislo (Zdroj: vlastní)

```
def Nenicislo(vstup1,vstup2):
    if not ((vstup1.isnumeric()) or (vstup2.isnumeric())):
        messagebox.showerror("Upozornění", "Ani jeden vstup není číselná hodnota!")
    elif  not (vstup1.isnumeric()):
        messagebox.showerror("Upozornění", "První vstup není číslo!")
    elif not (vstup2.isnumeric()):
        messagebox.showerror("Upozornění", "Druhý vstup není číslo!")
```

Obrázek 46 Ukázka funkce Nenicislo (Zdroj: vlastní)



Obrázek 47 Výsledek Kalkulačka Python (Zdroj: vlastní)

3.5.6 ÚLOHA 6

Popis úlohy

V této úloze je ukázáno importování *tkinteru* a dalších modulů. Je zde ukázáno vytvoření dialogového okna se vstupem nebo vygenerování náhodné hodnoty. Jsou zde vytvořeny widgety, kterým jsou přiřazeny různé parametry. Využíváme zde cyklu *while*, podmínek a globální proměnné.

Kód

Importujeme modul *tkinter*. Hlavnímu oknu nastavíme *title* a *geometry* a vytvoříme hlavní smyčku *mainloop*. V úloze potřebujeme dva *Labely*, které využijeme pro vypsání textu jako nadpisů. Přiřazujeme jim rodiče *root*. Dalším parametrem je *text*, jenž nabývá textové hodnoty. Samotný text je dále upravován přes parametr *font*, v němž je nastavený styl, velikost a tučné písmo. Dále jsou vytvořeny tři vstupy pro uživatele, kdy první dva jsou pro zadání nejnižší a nejvyšší možné hodnoty. Třetí je pro zadání tipovaného čísla. Všechny mají přiřazeného rodiče *root*. V práci jsou uvedeny tlačítka, kdy jim nastavujeme *root*, *text* a *command* s danou funkcí. Pro umístění widgetů je zde využita metoda *grid*, ve které jsou parametry *column* a *row*. U tlačítka *button_potvrdit* je nastaven parametr *padx*, kterým docílíme odsazení od ostatních prvků.

Nyní se přesuneme k jednotlivým funkcím. Jako první zde máme funkci *zamichat*. Tato funkce vygeneruje náhodné číslo v intervalu zadaný od uživatele. Deklarujeme proměnné *hodnota1* a *hodnota2*. Do první vložíme hodnotu z *entry_od* přes metodu *get* a do *hodnota2* přiřadíme hodnotu *entry_do.* Dále potřebujeme zjistit, zda proměnné jsou čísla. Docílíme toho podmínkou, ve které využijeme metodu *isnumeric* na obě proměnné. Je nutné, aby obě proměnné byly čísla, proto je zde operátor *and*. Pokud alespoň jedna z nich nebude číslo, podmínka se vyhodnotí jako *False* a zobrazí se *messagebox* s titulkem *"Upozornění"* a textem *"Nezadali jste celé číslo!"*. Jestliže se vyhodnotí jako *True*, tak se *hodnota1* a *hodnota2* přetypují na datový typ *int*. Máme čísla uložena v proměnných a potřebujeme nějakým způsobem mezi nimi vygenerovat číslo. K tomu můžeme využít modul *random*, ten ale nejprve je důležité importovat. Importujeme ho na začátku dokumentu. Zároveň je důležité si uvědomit, že vygenerované číslo budeme potřebovat i v jiných funkcích. Proto je zde využíváno globální proměnné *vysledek*. Proměnná je deklarována před funkcemi. Pokud chceme s touto proměnnou pracovat musíme na ni odkazovat klíčovým slovem *global*. Jinak bychom vytvořili lokální proměnnou s názvem *vysledek*. Máme vše připraveno pro vygenerování náhodného čísla. Do globální proměnné *vysledek* přiřazujeme hodnotu vygenerovanou přes metodu *randint* z modulu *random*, do které vkládáme proměnné *hodnota1* a *hodnota2*. Tento zápis vygeneruje číslo z uzavřeného intervalu.

```
def zamichat():
    global vysledek
    hodnota1= entry_od.get()
    hodnota2=entry_do.get()
    if hodnota1.isnumeric() and hodnota2.isnumeric():
        hodnota1=int(hodnota1)
        hodnota2=int(hodnota2)
        vysledek = random.randint(hodnota1, hodnota2)
    else:
        messagebox.showwarning("Upozornění", "Nezadali
```

Obrázek 48 Ukázka funkce zamichat (Zdroj: vlastní)

Přesouváme se k funkci *tipnout*. Na začátku funkce odkazujeme na globální proměnnou *vysledek*. Abychom mohli evidovat počet pokusů od uživatele je nutné deklarovat lokální proměnnou *pokus* a přiřazujeme ji číslo 1. Dále si deklarujeme proměnnou *vyhra* a *odhad*. Proměnné *vyhra* přiřazujeme *True* a do *odhad* hodnotu, která je v *entry_tip*. Poslední

proměnnou je *vyherni_cislo*, do které vkládáme hodnotu globální proměnné *vysledek*. Musíme vytvořit podmínku, ve které se ptáme na proměnnou *odhad* metodou *isnumeric*. Při vyhodnocení *False* se zobrazí *messagebox* s textem *"Je třeba zadat číslo!"*. Pokud se podmínka splní následuje přetypování *odhad* na *int*. Abychom se uživatele opakovaně dotazovali na nový tip, využijeme zde cyklu *while*. Podmínkou cyklu určujeme, že blok kódu bude probíhat do doby, kdy se *odhad* nerovná *vyherni_cislo*. Jinak řečeno tip od uživatele nebude stejný jako výherní číslo. Pokud *odhad* není stejný jako *vyherni_cislo*, tak chceme uživateli nějakým způsobem napovědět. Pro vytvoření nápovědy využijeme proměnné *odhad* a zeptáme se, jestli je větší nebo menší než *vyherni_cislo*. To řešíme větvenou podmínku, ve kterých používáme operátory „*>*“ nebo „*<*“. V blocích kódu každé zmíněné podmínky je deklarovaná proměnná *novy_odhad*, do které chceme vytvořit dialogové okno se vstupem. Pro tento typ oken je v *tkinteru* modul s názvem *simpledialog*, kterou musíme importovat podobně jako *messagebox*. Jakmile máme modul importován, do proměnné *novy_odhad* chceme přiřadit nový tip. To je možné přes *simpledialog* s funkcí *askinteger*. Tuto funkci využíváme z důvodu toho, že od uživatele chceme zadat celé číslo. Kdyby ho nezadal, zobrazí se nám zpráva, která uživatele upozorní, že zadaná hodnota není celé číslo. Pokud hodnota je celé číslo, tak se přiřadí do proměnné *novy_odhad*. Ve funkci *askinteger* využíváme parametry *title*, *prompt* a *parent*. Do prvního přiřazujeme titulek. V *promptu* je hodnota neboli text, jenž chceme uživateli sdělit. Můžeme si všimnout, že je zde zřetězení textu a proměnné *odhad*. Tu musíme přetypovat na *str*, protože parametr *prompt* nabývá textové hodnoty. Poslední parametr je *parent*, kterému přiřazujeme *root*. Tímto parametrem dialogové okno zobrazujeme nad hlavním oknem. Dále v obou podmínkách se do proměnné *odhad* přiřazuje *novy_odhad*. Poslední proměnná je *pokus*, do které přiřazujeme hodnotu pokusů o jedna větší. Je zde ale možnost, kdy uživatel nebude chtít v hádání pokračovat a v dialogovém okně klikne na tlačítko „*Cancel*“. Pokud se tak stane, tak do proměnné *novy_odhad* se přiřadí hodnota *None*. Proto musíme větvenou podmínku rozšířit o další podmínku, která bude na prvním místě. V této podmínce se dotazujeme, zda *odhad* je *None*. Pokud je vyhodnocena jako pravda, proměnné *odhad* přiřazujeme hodnotu *vyherni_cislo* a proměnné *vyhra False*. *Odhad* jsme nastavili na tuto hodnotu z důvodu, aby při opětovném procházením se podmínka cyklu nesplnila. S proměnnou *vyhra* budeme pokračovat dále v kódu.

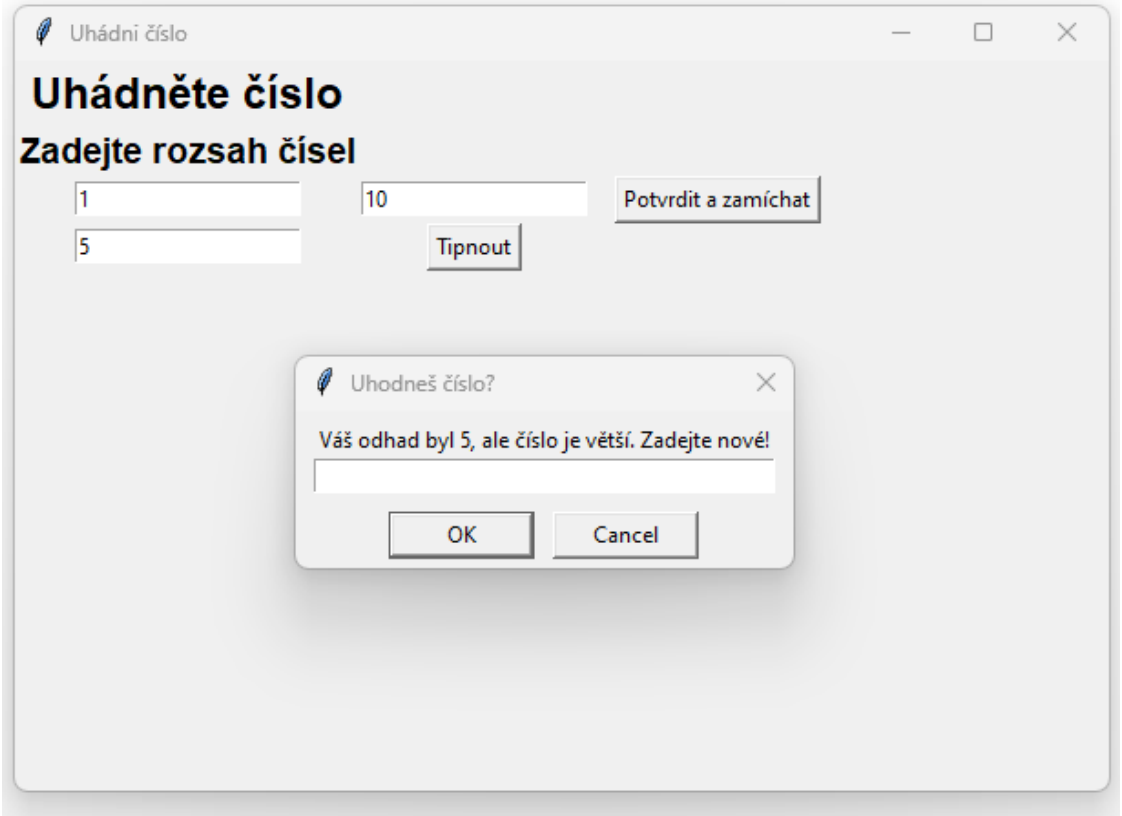
```
while(odhad!=vyherni_cislo):
    if(odhad is None):
        odhad=vyherni_cislo
        vyhra=False

    elif (odhad<vyherni_cislo):
        novy_odhad=simpledialog.askinteger(title="Uhodneš číslo?" ,prompt="
        odhad=novy_odhad
        pokus=pokus+1

    elif(odhad>vyherni_cislo):
        novy_odhad=simpledialog.askinteger(title="Uhodneš číslo?" ,prompt="
        odhad=novy_odhad
        pokus=pokus+1
```

Obrázek 49 Ukázka while a podmínek (Zdroj: vlastní)

Jako poslední věc, kterou je ve funkci nutné vyřešit je vyhodnocení hádání čísla. Využíváme zde podmínku, ve které se dotazujeme, jestli proměnná *vyhra* je *True*. Pokud ano, zobrazí se *messagebox* sdělující výhru s vypsaným počtem *pokusů* (nutné přetypovat z *int* na *str)*. Jinak se zobrazí *messagebox* ohlašující prohru a vypíše nám *vysledek*.



Obrázek 50 Výsledek Uhádněte číslo Python (Zdroj: vlastní)

3.5.7 ÚLOHA 7

Popis úlohy

V úloze Auta je importován modul *tkinter*, pomocí které je vytvořené hlavní okno. Jsou zde využívány různé widgety, kterým jsou nastavovány parametry a umístění přes dvě různé metody. V úloze je pro ukládání hodnot využito *listu* a slovníku. Dále jsou zde funkce, ve kterých je využíváno přetypování hodnot, cyklů a podmínek.

Kód

Importujeme moduly a vytvoříme hlavní okno a nastavíme mu šířku a výšku přes metodu *geometry* a titulek přes *title*. Nezapomeňme na vykreslení smyčky *mainloop*. Nyní si definujeme jednotlivé widgety. Úloha obsahuje *Entry*, *Buttony* a *Text*. Všem widgetům nastavujeme rodiče *root*. Do *Buttonů* navíc ještě přiřazujeme parametr *text* a *command*. Na widgetu *Text* kromě rodiče přiřazujeme navíc šířku a výšku. Umístění všech widgetů kromě *Text*, je provedeno pomocí metody *grid*. V ní nastavujeme hodnotu řádkům a sloupečkům. Samotný widget *Text* umisťujeme přes metodu *place*.

Nyní se dostáváme k logické části úlohy. Jelikož chceme vypisovat všechna auta, která uživatel zadal, je pro jejich uložení nutné deklarovat prázdný *list*. Zároveň chceme autu přiřazovat více „vlastností“ (značka, model, hmotnost) a následně ho i s nimi uložit na určitou pozici v *listu*. K tomu je zde vytvořena funkce s názvem *auto* s parametry *vstup_znacka*, *vstup_model* a *vstup_hmotnost*. Uvnitř této funkce je nutné vytvořit slovník (*dictionary*), který se skládá ze složených závorek. Ve slovnících uvádíme klíče s hodnotami. V našem případě definujeme první klíč s názvem *"znacka"*, kterému dvojtečkou přiřazujeme hodnotu *vstup_znacka* neboli jeden z parametrů funkce. Pomocí klíče je možné přistupovat k daným hodnotám. V poslední řadě musíme pomocí klíčového slova *return* vrátit hodnotu z proměnné *auta*.

```
def auto(vstup_znacka,vstup_model,vstup_hmotnost):
    auta= {"znacka":vstup_znacka,"model":vstup_model,"h
    return auta
```

Obrázek 51 Ukázka funkce auto (Zdroj: vlastní)

Dále zde máme funkci *pridat*, ve které je podmínka kontrolující vyplnění vstupů od uživatele. Pokud se podmínka splnila musíme vytvořit další podmínku, tentokrát kontrolujeme, zda vstup pro hmotnost je číslo. Při splnění deklarujeme proměnné

vstup_znacka, *vstup_model* a *vstup_hmotnost*. Každé přiřazujeme hodnotu jednotlivých vstupů metodou *get*. Následuje deklarace proměnné *auticko*, ve které je volána funkce *auto*, kdy její argumenty jsou vytvořené proměnné. Návratovou hodnotu vkládáme do *listu* pomocí metody *append*. Před samotným vypisováním je důležité widget *textarea_vypis* smazat pomocí metody *delete*. Pro vypisování hodnot slovníků je zde využíván cyklus *for*. Vytvoříme proměnnou *car*, díky které budeme postupně procházet jednotlivé položky v *auta_list*. Pro každý slovník se provede blok kódu, ve kterém do *textarea_vypis* pomocí metody *insert* vkládáme jeho hodnoty. Prvním parametrem určujeme vypisování za poslední znak widgetu. V dalším parametru je malé *f* neboli formátované řetězcové literály. Ten využíváme pro vkládání hodnot jednotlivých klíčů konkrétního slovníku (např. *{car["znacka"]}*) do řetězce. Před hodnotami je text (např. *Značka:*). Na konci řetězce je *"\n"*, což je kombinace znaků, s nímž docílíme odřádkování.

```
for car in auta_list:
    textarea_vypis.insert("end",f'Značka: {car["znacka"]},
```

Obrázek 52 Ukázka cyklu a vkládání hodnot metodou insert (Zdroj: vlastní)

Dostáváme se k výpočtu celkové hmotnosti. Funkce má název *weight*. Deklarujeme v ní proměnnou *vypis* a přiřazujeme ji hodnotu nula. Následuje cyklus *for*, kdy procházíme jednotlivé slovníky v *listu*. V cyklu sčítáme číselné hodnoty hmotnosti do proměnné *vypis*. Pro sčítání hmotností uvádíme název klíče ("*hmotnost*") do hranatých závorek proměnné *pozice*. Celý tento řetězec přetypujeme na datový typ *int*. Poté vymažeme *entry_celkovaHmotnost* od začátku do konce přes metodu *delete*. Hodnotu proměnné *vypis* vložíme do *entry_celkovaHmotnost*. Hodnotu v proměnné *vypis* chceme využít i ve funkci *souhrn*, proto využíváme *return*.

```
vypis+=int(pozice["hmotnost"])
```

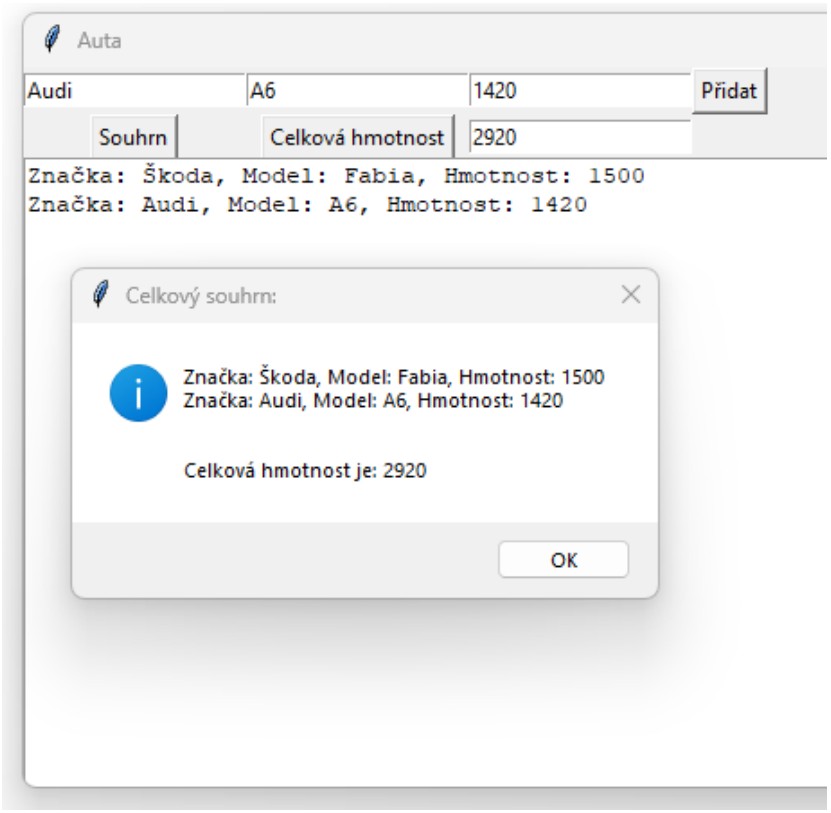
Obrázek 53 Ukázka proměnné vypis (Zdroj: vlastní)

Poslední funkce s názvem *souhrn* zobrazuje dialogové okno, ve kterém jsou vypsané hodnoty slovníků uložených v *auta_list*. Deklarujeme v ní proměnné *znacka*, *model* a *hmotnost*, do nich přiřazujeme hodnoty daných *Entry*. Dále využíváme podmínky, ve které se ptáme, jestli jednotlivé proměnné mají hodnotu prázdného textu. Využíváme operátoru *and*. Pokud ani jeden není prázdný deklarujeme proměnnou *vypsani* a přiřazujeme ji hodnotu *textarea_vypis*. Do souhrnu chceme vypsat i celkovou hmotnost.

Proto deklarujeme proměnnou *celkovahmotnost* a přiřazujeme ji návratovou hodnotu z funkce *weight*. Poté vytvoříme *messagebox*, do kterého řetězíme proměnnou *vypsani*, znak pro odřádkování (*"\n"*), text "*Celková hmotnost je:* " a přetypovanou hodnotu *celkovahmotnost* na *str*. Pokud některá z předešlých podmínek nebyla splněna, bude zobrazen *messagebox* s daným upozorněním.

```
if znacka and model and hmotnost:
    vypsani=textarea_vypis.get("1.0","end")
    celkovahmotnost= weight()
    messagebox.showinfo("Celkový souhrn:", vypsani +"\n" +"Celková h
```

Obrázek 54 Ukázka bloku kódu podmínky funkce souhrn (Zdroj: vlastní)



Obrázek 55 Výsledek Auta Python (Zdroj: vlastní)

3.5.8 ÚLOHA 8

Popis úlohy

V této úloze se zaměříme na *Canvas*, ve kterém bude vykreslena kulička. Dále je zde ukázáno pomocí podmínek nastavení hranic pro odrážení kuličky. Pohyb kuličky se spouští přes tlačítko.

Kód

Nejprve je nutné importovat modul *time* a *tkinter*, přes který vytvoříme hlavní okno. Dále si deklarujeme proměnnou *kanvas*, do které přes *tkinter* vytvoříme *Canvas* s parametry *root*, rozměry a ohraničením, na který následně voláme metodu *pack*. Pro vytvoření koule je nutné využít metodu *create_oval*. Do této metody přiřadíme parametry pro levý horní roh, pravý dolní roh a barvu kuličky. Dále si deklarujeme globální proměnné *prvni* a *druha*, do kterých přiřadíme číselnou hodnotu. Tyto proměnné budeme přiřazovat jako argumenty do metody pro rozpohybování kuličky. Vytvoříme funkci *pohyb*, kdy pro měnění hodnot v globálních proměnných je nejprve musíme uvést slovem *global*. Dále si deklarujeme proměnné, do kterých budeme ukládat konkrétní souřadnice (*x1*, *y1*, *x2*, *y2*) kuličky. Využíváme metodu *coords* a jako argument vkládáme proměnnou *koule*. Následují podmínky, ve kterých se ptáme, zda se jednotlivé souřadnice rovnají pozicím, určující hranici *kanvas*. V první podmínce se dotazujeme, jestli hodnota *y2* je rovna šířce *kanvas*, od které odečítáme číslo čtyři. Číslo odečítáme z důvodu, že i přes nastavení šířky je reálná hodnota o čtyři větší. Při splnění podmínky od proměnné *druha* odečítáme hodnotu a výsledek odčítání přiřazujeme do proměnné *druha*. Tím docílíme inverze směru vykreslování. V další podmínce se dotazujeme, zda hodnota *y1* je rovna číslu pět. Při splnění podmínky se k *druha* přičte pět a tento výsledek přiřazujeme do proměnné *druha*. Stejný zápis platí i pro zbylé podmínky, kdy se využívají jiné proměnné (*x2*, *x1* a *prvni*).

```
if y2 == kanvas.winfo_height()-4:
    druha=druha-5
if y1 == 5:
    druha=druha+5
if x2 == kanvas.winfo_width()-4:
    prvni=prvni-5
if x1 == 5:
    prvni=prvni+5
```

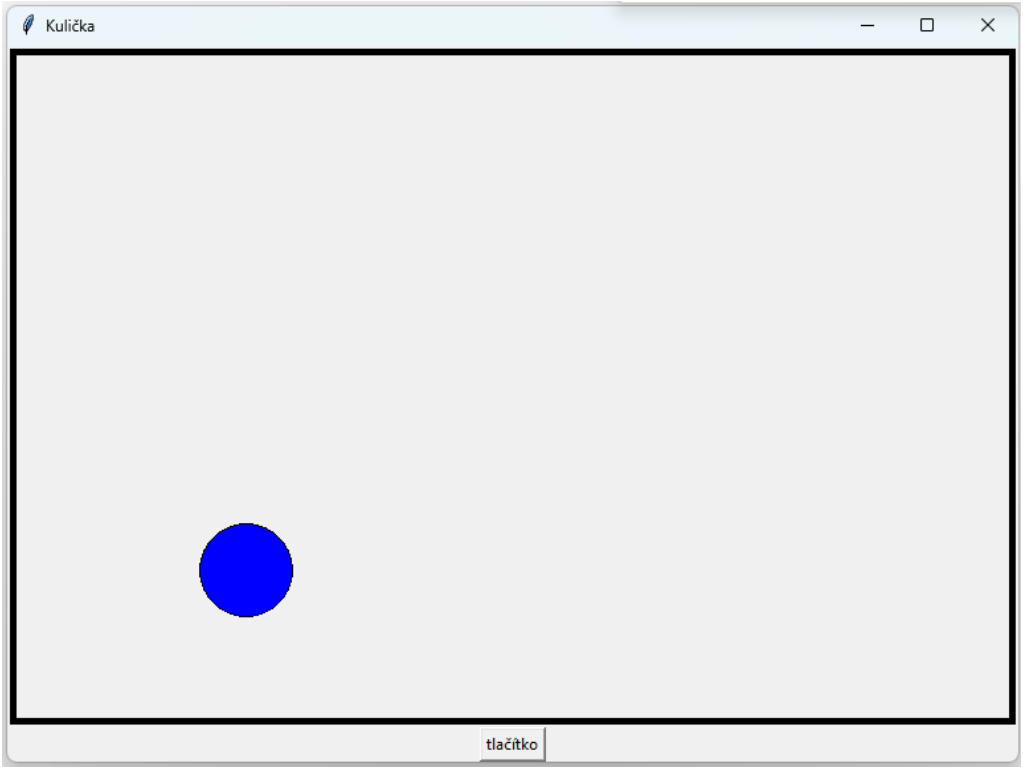
Obrázek 56 Ukázka zápisu podmínek (Zdroj: vlastní)

Následuje využití metody *move*, pomocí které rozpohybujeme kuličku. Do jejích parametrů přiřadíme proměnnou *koule*, tímto parametrem uvádíme, co se vykreslí na *Canvas*. Do druhého a třetího parametru uvádíme na jakou pozici chceme kuličku posunout. Dále na *kanvas* voláme metodu *after*. Tou voláme funkci *pohyb* po uplynutí zadaného času. Zatím jsme si neuvedli, k čemu ve funkci *pohyb* je využita funkce *sleep* z modulu *time*.

Pokud by *sleep* nebyla v *pohyb* uvedena, *Canvas* by reagoval na kurzor a kulička by po přejetí myší zrychlila.

```
kanvas.move(koule,prvni,druha)
kanvas.after(5,pohyb)
```

Obrázek 57 Ukázka metody move a after (Zdroj: vlastní)



Obrázek 58 Výsledek Kulička Python (Zdroj: vlastní)

3.5.9 ÚLOHA 9

Popis úlohy

V této úloze využíváme *Canvas* pro vykreslení sloupcového grafu. Pro vypisování hodnot jsou použity *Labely*. Dále je v úloze ukázáno využití cyklů *for* a *listu*. U Canvasu se využívají různé metody, kterými vykreslujeme sloupečky nebo linii vyznačující průměrnou hodnotu.

Kód

Je nutné importovat *tkinter*, vytvoříme hlavní okno, kterému nastavíme titulek a rozměry. Nutné je spustit hlavní smyčku. Dále vytvoříme *Canvas* a přiřazujeme rodiče *root* a nastavíme mu výšku a šířku. Abychom změnili barvu pozadí voláme na proměnnou *kanvas* metodu *configure*, v níž je parametr *bg* nastaven na tyrkysovou barvu. *Canvas* pomocí metody *grid* umístíme na druhý řádek a nultý sloupeček. Pro nadpis a vypisování

hodnot je využito *Labelů*. V prvním z nich se bude zobrazovat text *„Náhodný graf“*, který má nastavený *font*. Další widget, jenž je využíván je *Button*. Tlačítku nastavíme rodiče *root*, parametry *text* a *command*. Nyní si všechny widgety musíme umístit, využijeme k tomu metodu *grid*. Dále využijeme parametru *sticky* s hodnotou *"w"*.

Nyní se přesouváme k jediné funkci úlohy s názvem *start*. Vytvoříme prázdný *list* s názvem *listhodnot*. Pro vygenerování náhodných hodnot využijeme modul *random*, který importujeme na začátku dokumentu. Pro generování používáme cyklu *for,* ve kterém přes funkci *range* stanovíme počet opakování. Argumentem je číslo osm, to znamená že cyklus proběhne osmkrát. Vně cyklu pomocí modulu *random* a metody *randint* vygenerujeme celé náhodné číslo v daném rozsahu včetně krajních hodnot. Vygenerovaná čísla uložíme do *listhodnot* metodou *append*. Tímto zápisem je do *listu* postupně uloženo osm hodnot. Dále si musíme uložit maximální a minimální hodnotu. Proto si deklarujeme proměnnou *nejvyssi_hodnota* a *nejnizsi_hodnota*. V nich uvádíme buď funkci *max* nebo *min*. Tyto funkce vyberou největší nebo nejmenší hodnotu z *listu*. Proto jim jako argument vkládáme *listhodnot*.

```
nejvyssi_hodnota = max(listhodnot)
nejnizsi_hodnota = min(listhodnot)
```

Obrázek 59 Ukázka maximální a minimální hodnoty (Zdroj: vlastní)

Pro vytvoření sloupečků je důležité určit počáteční pozici vykreslování. K tomu je zde deklarovaná proměnná *x*, do které přiřazujeme číselnou hodnotu. Následuje cyklus *for*, ve kterém se přes podmínky dotazujeme, zda hodnota na dané pozici v *listu* je *nejvyssi_hodnota* nebo *nejnizsi_hodnota*. Aby bylo možné procházet jednotlivé pozice, před cyklem deklarujeme proměnnou *y* s hodnotou 0. Využijeme ji jako index pro procházení jednotlivých prvků v *listu*. Uvnitř cyklu je zmíněná větvená podmínka. V první se dotazujeme, jestli hodnota v *listhodnot* na pozici *y* je stejná jako *nejvyssi_hodnota*. Pokud je to pravda, na proměnné *kanvas* se zavolá metoda *create_rectangle*, která se používá pro vykreslení obdélníkového tvaru. Parametry pro vykreslení jsou čtyři. První dva reprezentují levý horní roh a zbylé dva pravý dolní roh obdélníku. Dále jsou dobrovolné parametry, v nich je možné specifikovat např. vzhled. Prvním argumentem je proměnná x, druhému určujeme ypsilonovou souřadnici neboli výšku sloupce. Proto si pod *listhodnot* deklarujeme proměnnou *vyska_kanvas*, které

je přiřazena výška *Canvasu* metodou *winfo_height*. Tuto hodnotu musíme přetypovat na typ *int*, protože s ní budeme provádět aritmetickou operaci (odčítání).

```
vyska_kanvas=int(kanvas.winfo_height())
```

Obrázek 60 Ukázka proměnné vyska_kanvas (Zdroj: vlastní)

Vytvořenou proměnnou vkládáme do druhého parametru a odečítáme od ní hodnotu z *listhodnot* na pozici *y*. Jako třetí argument vkládáme proměnnou *x* a k ní připočítáváme číslo 50. Jinými slovy udáváme šířku sloupečku. V posledním parametru je proměnná *vyska_kanvas*. Tím docílíme že pravý dolní roh se nachází na spodní hraně *Canvasu*. Tímto zápisem se nám vykreslí bezbarví sloupeček. Abychom ho obarvili je nutné využít ještě jeden parametr a tím je *fill* s názvem dané barvy.

```
kanvas.create_rectangle(x,vyska_kanvas-listhodnot[y],x+50,50,fill="green")
```

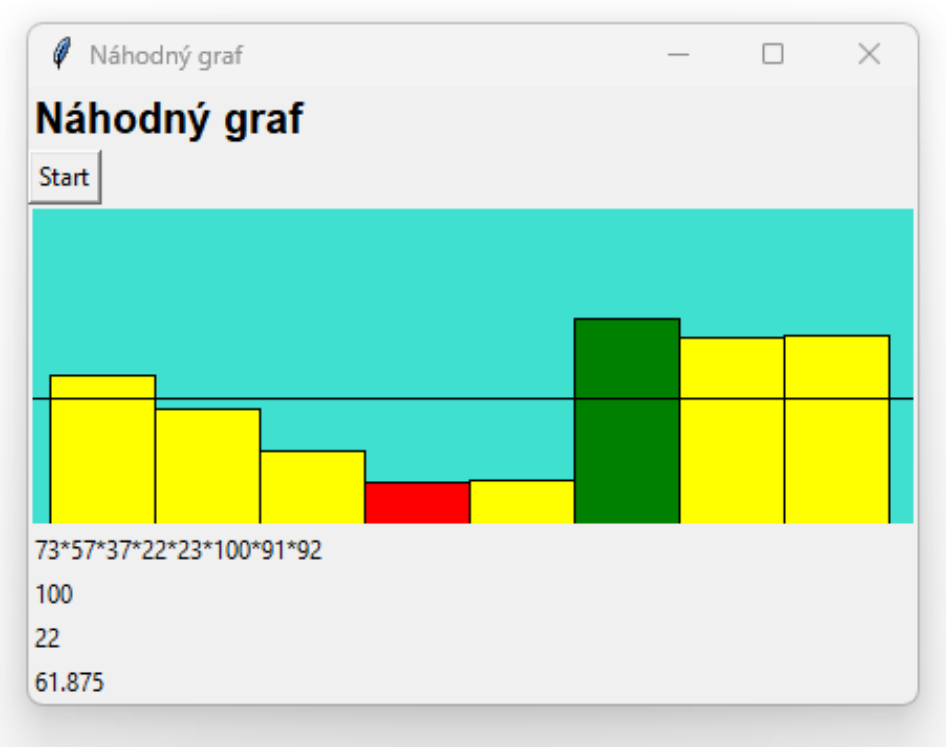
Obrázek 61 Ukázka vytvoření sloupečku (Zdroj: vlastní)

Tento zápis opakujeme ve všech následujících podmínek cyklu, s rozdílem barvy. V případě, že hodnota v *listu* je největší, nastaví se barva zelená. Pokud je nejmenší, tak červená jinak žlutá. Po vyhodnocení podmínek se do proměnné *y* přičítá číslo 1. Je to z důvodu postupného procházení všech prvků. Do proměnné *x* připočítáváme číslo 50 neboli posouváme počáteční pozici o šířku sloupce. Proto se nám vykreslují sloupečky vedle sebe. Nyní se dostáváme k vypisování největší, nejmenší a průměrné hodnoty z *listu*. Mezi jednotlivé prvky z *listu* chceme vypisovat znak hvězdička (*). Deklarujeme proměnnou *vypis*, do které přiřazujeme znak hvězdičky a voláme metodu *join*. Pro vložení hvězdičky mezi jednotlivé prvky je zde využito cyklu *for*. Následně řetězec uložený v proměnné *vypis* se vkládá do parametru *text* widgetu *label_hodnoty*. Nyní si vypíšeme do *Labelů* nejvyšší a nejmenší hodnotu. Na dané *Labely* zavoláme metodu *config*, ve které do parametru *text* vložíme proměnné *nejvyssi_hodnota* nebo *nejmensi_hodnota*. Nesmíme zapomenout přetypovat na datový typ *str*. Abychom mohli vypsat aritmetický průměr, tak ho musíme nejdříve vypočítat. Deklarujeme proměnnou *soucet*, ve které pomocí funkce *sum* sečteme všechny hodnoty z *listhodnot*. Následuje deklarace proměnné *prumer*. V ní provádíme podíl mezi *soucet* a délkou *listhodnot*. Délku zjistíme pomocí funkce *len*. Pro vypsání proměnné *prumer* do *Labelu* je nutné přetypovat na *str*. V grafu chceme průměrnou hodnotu vyznačit linkou, která povede přes všechny sloupce. Proto musíme na proměnné *kanvas* volat

metodu *create_line*. Tato metoda má stejné parametry jako *create_rectangle*. V prvním parametru uvádíme číslo 0, protože chceme začít vykreslovat od levé hrany. Výšku, ve které se linie má začít vykreslovat dosáhneme rozdílem mezi proměnnou *vyska_kanvas* a *prumer*. Do třetího parametru je přiřazena šířka *Canvasu*, jenž získáme zavoláním metody *winfo_width*. Do posledního parametru znovu uvádíme hodnotu rozdílu mezi *vyska_kanvas* a *prumer*.

```
soucet=sum(listhodnot)
prumer=soucet/len(listhodnot)
label_prumer.config(text=str(prumer))
kanvas.create_line(0,vyska_kanvas-prumer,kanvas.winfo_width(),vyska_kanvas-prumer)
```

Obrázek 62 Ukázka výpočtu průměru a vykreslení linie (Zdroj: vlastní)



Obrázek 63 Výsledek Grafy Python (Zdroj: vlastní)

3.6 ÚLOHY VYTVOŘENY S VYUŽITÍM MODULŮ

V následujících úlohách si ukážeme řešení za využití externích modulů. Moduly v Pythonu zrychlují a mnohdy zjednodušují řešení mnoha problémů. Úlohy v předmětu PGM1P jsou v některých případech natolik základní, že využitím modulů se jejich řešení nijak nestane jednodušším, ani nepřinese nic nového. Proto jsme vybrali a s moduly realizovali pouze ty úlohy, kde došlo ke zjednodušení řešení, nebo ke zlepšení jejich funkcionality.

3.6.1 ÚLOHA 1

Popis úlohy

První úloze je předělána část úlohy Galerie pomocí modulu PIL. Využíváme ji zde pro vykreslení obrázku gif. Tento modul není součástí standartní verze Pythonu a je nutné ho nainstalovat. Pro grafické rozhraní je využit modul *tkinter*.

Kód

Importujeme *tkinter* a vytvoříme rodiče *root*, kterému nastavíme *geometry* a *title*. Nezapomeňme vytvořit a nadefinovat widgety, jako v úloze bez modulů. V této části kódu z modulu PIL importujeme *Image*, *ImageTK* a *ImageSequence*. S jejich pomocí dokážeme vykreslit gif soubor. Dále si deklarujeme proměnnou *picture*, do které přes *PhotoImage* parametrem *file* přiřadíme obrázek. Začneme funkcí s názvem *obrazek1*, přes kterou do *label_Obrazky* a metodou *config* s parametrem *image* přiřazujeme *picture*. Nyní přecházíme k funkci, kterou budeme vykreslovat gif. Název této funkce je *pohyb_gif*. Zde deklarujeme proměnnou *gif*, do které přes funkci *open* modulu *Image* vkládáme jako argument *giphy.gif*. Dále využíváme *for* cyklu, kde deklarujeme proměnnou *cast_gif* a *ImageSequence.Iterator*, což je funkce vracející iterátor z *gif*. Jinak řečeno *cast_gif* je jeden iterátor ze sekvence neboli jeden obrázek, z kterého se postupně skládá animace. Vně cyklu dochází k samotnému vykreslování obrázků, do proměnné *gif* se přiřazuje instance třídy *PhotoImage* neboli se ukládá jeden obrázek ze sekvence. Poté se vkládá do *label_Obrazky* a *root* přes funkci *update* se překreslí. Tím docílíme viditelné měnění obrázku v *Labelu*. Aby se vykreslení *gif* neprovedlo příliš rychle je zde využit modul *time* s funkcí *sleep*, kterou nastavíme pauzu mezi jednotlivými obrázky. Tento modul je nutné importovat.

```
gif = Image.open("obrazky_galerie\giphy.gif")
for cast_gif in ImageSequence.Iterator(gif):
    gif = ImageTk.PhotoImage(cast_gif)
    label_Obrazky.config(image=gif)
    root.update()
    time.sleep(0.04)
vykreslit = False
```

Obrázek 64 Ukázka vykreslování gifu (Zdroj: vlastní)

Než si rozebereme funkci *zmena*, je důležité si uvědomit, že potřebujeme rozlišit, kdy se obrázek nebo gif budou vykreslovat. Proto je pod proměnnou *picture* vytvořena další

proměnná *vykreslit* s hodnotou *True*. Pro upřesnění, pokud hodnota proměnné je *True*, je možné vykreslit gif, jestliže je *False* může být vykreslen *picture*. Této proměnné budeme měnit hodnotu na základě kliknutého tlačítka. Ve funkci *zmena* přes klíčové slovo *global* jsou uvedeny proměnné *gif* a *vykreslit*. Následnými podmínkami ověřujeme hodnotu proměnné *vykreslit*. V první se ptáme, jestli vykreslit nabývá hodnoty *False*. Abychom dosáhli takové situace musíme proměnné *vykreslit* za cyklem *for* ve funkci *pohyb_gif* nastavit hodnotu na *False*. Tím docílíme splnění podmínky neboli po vykreslení gifu, může být hodnota v *Labelu* změněna na *picture*. Pokud se nesplní první podmínka, tak se dotazujeme, jestli hodnota *vykreslit* je *True*, pokud tomu tak je, zavoláme funkci *pohyb_gif*.

Obrázek 65 Výsledek Galerie Python s externími moduly

3.6.2 ÚLOHA 2

Popis úlohy

V této úloze je předělaná úloha s názvem Kalkulačka. Je zde využit modul *numpy*, kterou používáme pro výpočet logaritmů, mediánu a směrodatné odchylky. Základ této úlohy zůstává stejný (základní aritmetické operace), proto se zde zaměříme pouze na nově vytvořené funkce. *Numpy* je nutno nainstalovat.

Kód

Importujeme *tkinter* a *numpy*. Pro nové matematické operace vytvoříme nové *Buttony* s rodičem *root*. Dále přiřadíme parametry *text* a *command*. Následně tlačítka umístíme

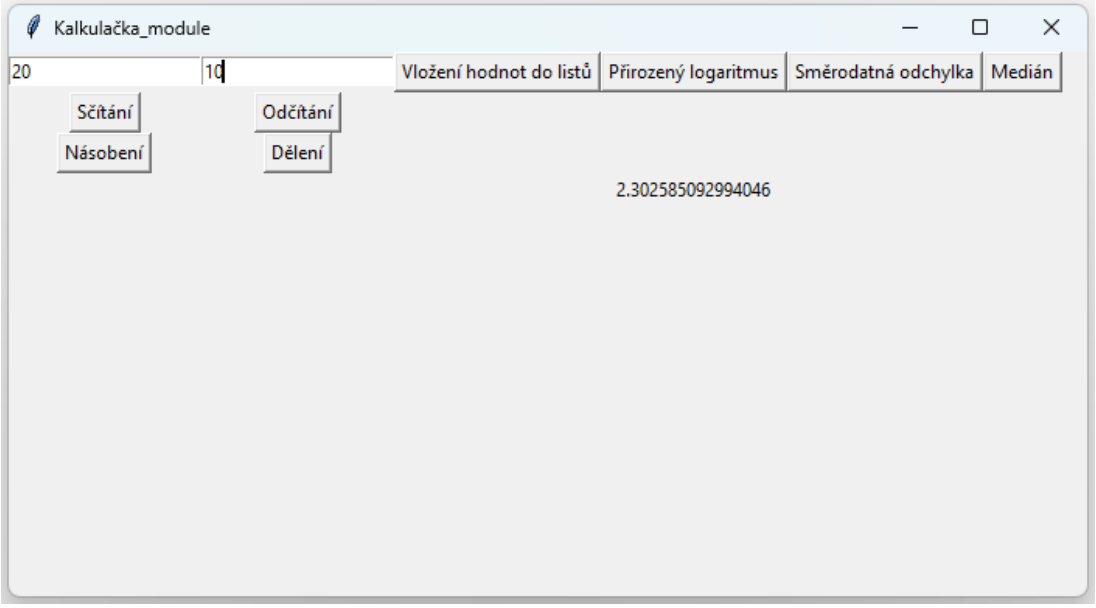
přes *grid* s parametry řádek a sloupec. Nyní se dostáváme k popisu funkcí. Jako první si rozebereme funkci s názvem *vlozenihodnot*. Přes tuto funkci budeme hodnotu v *label_vysledek* ukládat do nového *listu* s názvem *listhodnot*. Hodnoty uložené v *listu* budou využity pro výpočet mediánu a směrodatné odchylky. Nejprve se v podmínce ptáme, zda *label_vysledek* nabývá nějaké hodnoty. Pokud to tak je, deklarujeme proměnnou a přiřazujeme ji hodnotu *Labelu*. Tuto hodnotu dále vkládáme do *listhodnot*. Další funkce má název *logaritmy*. V ní deklarujeme proměnnou a přiřazujeme ji hodnotu z *label_vysledek*. Poté vytvoříme podmínku, ověřující, zda *Label* není prázdný. Při kladném vyhodnocení uvádíme další podmínku. Tentokrát se dotazujeme, jestli hodnota *Labelu* přetypovaná na *float* je větší než nula. Přetypování je zde z důvodu možného zadání desetinného čísla. Pokud i tato podmínka je kladně vyhodnocena, tak do nově deklarované proměnné přiřazujeme hodnotu logaritmu, který je získán přes funkci *log*. Následuje funkce *odchylka*, kdy využijeme modulu *numpy* a vypočítáme směrodatnou odchylku z hodnot uložených v *listu*. Uvádíme podmínku, ve které kontrolujeme, jestli *list* není prázdný. Pokud je vyhodnocení *True*, vytváříme proměnnou *smerodatnaodchylka*. Přiřazujeme ji hodnotu získanou přes funkci *std* z modulu *numpy* s argumentem *listhodnot*. Tím vypočítáme směrodatnou odchylku, kterou vypisujeme do *label_vysledek*. Další funkcí je *median*. Touto funkcí počítáme medián, kdy kód funkce je podobný jako u funkce *odchylka*. Je zde využíváno stejné podmínky, v níž se ptáme, zda je *list* prázdný. Pokud vyhodnocení je *True*, tak vytvoříme proměnnou *hodnota_median* a přes funkci *median* s argumentem *listhodnot* dostaneme výsledek. Zmíněnou proměnnou přiřazujeme *label_vysledek*.

```
if(not listhodnot == []):
  smerodatnaodchylka = numpy.std(listhodnot)
```

Obrázek 66 Ukázka směrodatné odchylky (Zdroj: vlastní)

```
if(not listhodnot == []):
  hodnota_median=numpy.median(listhodnot)
```

Obrázek 67 Ukázka mediánu (Zdroj: vlastní)



Obrázek 68 Výsledek Kalkulačka s externími moduly

3.6.3 ÚLOHA 3

Popis úlohy

Pro rozpohybování kuličky je využito modulu *pygame*. Po spuštění aplikace se kulička rozpohybuje. Když narazí na hrany hlavního okna, tak se odrazí pod daným úhlem. Dále se zde nachází tlačítko, které je také vytvořeno pomocí *pygame*. Po jeho stisknutí se pohyb zrychlí. Před začátkem vytváření kódu je nutné nainstalovat *pygame*.

Kód

Jako první věc musíme samotný p*ygame* importovat. Dále pomocí funkce *init* inicializujeme p*ygame*. Poté si deklarujeme dvě proměnné pro šířku a výšku, přiřadíme jim hodnoty. Dále vytvoříme proměnnou *displej*, do níž přes funkci *set_mode* nastavujeme rozlišení okna. Jako argumenty použijeme proměnné *sirka* a *vyska*. Dále si vytvoříme titulek přes funkci *set_caption* a do parametru přiřazujeme textový řetězec. Dále deklarujeme proměnnou *tlacitko*, do které vytvoříme obdélník pomocí třídy *Rect* a nastavíme ji pozice obdelníka. Dále deklarujeme proměnné *barva_tlacitko*, *text_tlacitko* a *barva_text*. Obarvení tlačítka docílíme třídou *Color*, které přiřazujeme zelenou barvu. Do *text_tlacitko* přiřazujeme text tlačítka. V *barva_text* nastavujeme bílou barvu. Tyto proměnné později využijeme pro dosazení parametrů funkci *rect*, kterou využíváme u vykreslení obdélníku. Dále jsou deklarovány proměnné *radius*, *barva_kulicky*, *pozice_kulicka*, *x* a *y*. Zmíněným proměnným jsou přiřazeny hodnoty, kdy u *barva_kulicka* nastavujeme tři číselné hodnoty představující

barvu. Do *pozice_kulicka* vytváříme *list* o dvou hodnotách, které nám určují pozici vykreslení *kulicky*. Tyto hodnoty si uložíme do proměnných *x* a *y*, kdy do *x* přiřazujeme první a do *y* druhou pozici *listu*. Zmíněné proměnné jsou potřebné dále v kódu pro vykreslení. Dále si nastavíme rychlost, kterou chceme, aby se kulička pohybovala. Proto je zde vytvořena proměnná *speed*, do které je uložena počáteční rychlost. Tuto hodnotu si v inverzním tvaru ukládáme do proměnných *pozicex* a *pozicey*. Mínusem dosáhneme pohybu ke spodní hranici hlavního okna. Musíme si uvědomit, že pro pohyb k dolní hranici je nutné hodnoty snižovat, oproti pohybu nahoru, kdy je nutné přičítat. Nyní se dostáváme k vykreslení hlavní smyčky. Abychom kontrolovali, zda uživatel aplikaci neukončil, musíme vytvořit proměnnou *start* a nastavit ji hodnotu *True*. Dále definujeme cyklus *while*, který bude probíhat do té doby, dokud se *start* rovná *True*. V něm si dále vytvoříme další cyklus, tentokrát je to *for*, kdy *udalost* zastupuje konkrétní *pygame* událost. V podmínkách se dotazujeme, zda se uskutečnily události pro ukončení hlavního okna nebo kliknutí myši nad námi vytvořeným tlačítkem. Pro ukončení využíváme konstantu *QUIT*. Pokud uživatel vypnul hlavní okno tak se proměnná *start* nastaví na *False*. U druhé podmínky se ptáme, jestli se provedla událost kliknutí nějakého tlačítka myši nad vytvořeným tlačítkem. Konkrétně je zde využita konstanta *MOUSEBUTTONDOWN* a na proměnné *tlacitko* metoda *collidenpoint*, které jako argument nastavujeme *udalost.pos*. Pomocí *pos* jsme schopni získat pozici kurzoru. Pokud uživatel klikne na zmíněné tlačítko, tak se zavolá funkce *zrychleni*. Tuto funkci si rozebereme až po popisu samotného pohybu kuličky. Pro rozpohybování kuličky využíváme již deklarovaných proměnných *x* a *y*. K nim přičítáme hodnoty nacházející se v *pozicex* a *pozicey*. Následují podmínky, kdy v první kontrolujeme, zda *x* je menší než *radius* nebo je větší než *sirka* mínus *radius*. Touto podmínkou nastavujeme odrážení od levého a pravého okraje. Druhá podmínka je zapsána stejně, s rozdílem využití proměnné *y*. Druhou podmínkou kontrolujeme horní a spodní okraje. Pokud je některá z podmínek splněná, provede se inverze *pozicex* nebo *pozicey*, kterou docílíme změnění směru pohybu kuličky.

```
x += pozicex
y += pozicey
print(str(x) + "    " + str(y))
if x < radius or x > sirka - radius:
    pozicex = -pozicex
if y < radius or y > vyska - radius:
    pozicey = -pozicey
```

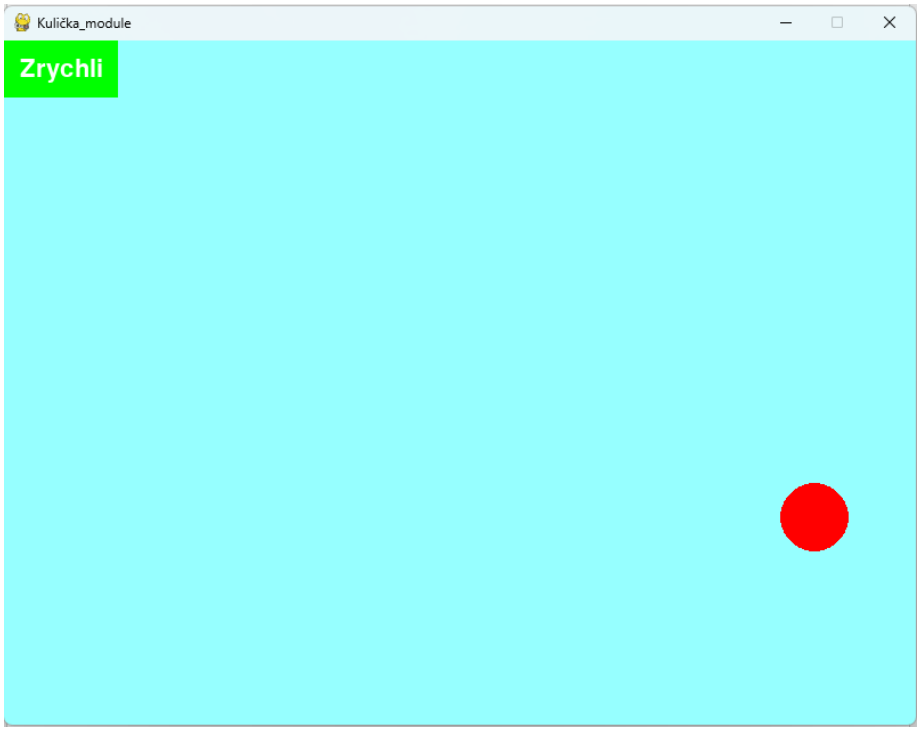
Obrázek 69 Ukázka kontroly pohybu kuličky (Zdroj: vlastní)

Kuličku je nutné vykreslit. Pro vykreslení využijeme funkci *circle*, do které přiřazujeme parametry *surface* (*displej*), barvu kuličky (*barva_kulicka*), souřadnice středu (*x* a *y*), které musejí být přetypovány na *int*. Posledním je poloměr (*radius*). Jelikož se kulička pouze vykresluje a ve skutečnosti se nepohybuje, musíme její cestu mazat. Proto před vykreslením použijeme zápis *displej.fill*, do které přiřadíme hodnoty pro tyrkysovou barvu. Dále musíme vykreslit obdélník reprezentující tlačítko. Pomocí funkce *rect* nelze přiřadit tlačítku text. Proto musíme vytvořit proměnnou *font* písma, která je deklarována před hlavní smyčkou. Pro vytvoření se využívá třídy *Font*, která má parametry pro typ fontu a velikost písma. Do typu fontu přiřadíme hodnotu *None*, ta nastaví výchozí font, jenž náš systém podporuje. Pokud bychom chtěli využít jiný, museli bychom ho externě instalovat. Dále vytvoříme text, který budeme vkládat do tlačítka. Toho docílíme přes metodu *render*. Do níž vložíme argumenty text_*tlacitko*, hodnotu *True*, která nám zajistí hladké hrany a proměnou *barva_text*. Nezapomeňme *font* uložit do nově deklarované proměnné *text*. Nyní musíme vytvořit obdélník s velikostí *text*, kterému přiřadíme pozici středu tlačítka. Jinak řečeno na *font* voláme metodu *get_rect*, která nám vrátí obdélník s rozměry *text* a pozicí tlačítka. Tento zápis přiřadíme proměnné *text_ctverec*. V posledním kroku musíme vykreslit *text* na *text_ctverec*. Toho docílíme přes metodu *blit*, které dosadíme argumenty *text* a *text_ctverec*. Pro průběžné vykreslování a překreslování se využívá funkce *flip*.

```
displej.fill((150, 255, 255))
pygame.draw.circle(displej, barva_kulicka, (int(x), int(y)), radius)
pygame.draw.rect(displej, barva_tlacitko, tlacitko)
text = font.render(text_tlacitko, True, barva_text)
text_ctverec = text.get_rect(center=tlacitko.center)
displej.blit(text, text_ctverec)
pygame.display.flip()
```

Obrázek 70 Ukázka vykreslení kuličky a „tlačítka“ (Zdroj: vlastní)

Nyní máme vytvořené tlačítko s vytvořenou událostí. Chybí nám vytvořit samotnou funkci *zrychleni*. Definujeme ji mimo hlavní smyčku. Vně funkce využíváme globálních proměnných *pozicex* a *pozicey*. Pro měnění zmíněných proměnných, musíme použít klíčové slovo *global*. Pro vytvoření zrychlení přičítáme do daných proměnných jejich aktuální hodnotu.



Obrázek 71 Výsledek Kulička s externími moduly

3.6.4 ÚLOHA 4

Popis úlohy

V této úloze vytvoříme sloupcové grafy přes *matplotlib*. Konkrétně využíváme modulu *pyplot*, kterým je možné grafy vykreslovat. Dále je zde využit modul *widgets*, z kterého importujeme *Button*. Pro tuto úlohu je nutné nainstalovat *matplotlib*.

Kód

V první části importujeme *pyplot* z *matplotlib*. Dále si importujeme z *widgets Button*. Musíme importovat modul *random,* budeme potřebovat generovat náhodná čísla. Nyní se dostáváme k deklaraci proměnných *fig* (*Figure*) a *ax* (*Axes*). Na *fig* přes různé metody volané na *ax* můžeme vykreslovat grafy, legendy apod. Pro nastavení rozměrů *fig* využíváme parametru *figsize* ve funkci *subplots*. Abychom mohli vykreslovat grafy, potřebujeme tlačítko. Deklarujeme proměnnou *pozicetlacitko*, do které přes funkci *axes*

s danými argumenty připravíme pozici tlačítka. Dále si vytvoříme proměnnou *tlacitko*, do které vytvoříme *Button* s argumenty *pozicetlacitko* a textem „*Start*“. Po nadefinování tlačítka deklarujeme *list* s názvem *data* a vkládáme počet sloupců, které chceme vykreslit. Důležité je zmínit, že nezávisí na hodnotách, ale na počtu prvků *listu*.

```
fig, ax = plt.subplots(figsize=(6, 8))
pozicetlacitko = plt.axes([0.01, 0.9, 0.1, 0.1])
tlacitko = Button(pozicetlacitko, 'Start')
data = [1, 2, 3, 4, 5, 6, 7, 8]
```

Obrázek 72 Ukázka fig, ax, tlačítka a listu (Zdroj: vlastní)

Dále si vytvoříme funkci s názvem *vykreslit*, v ní budeme vykreslovat sloupce s náhodnou hodnotou v daném intervalu (20 až 100). Jelikož uživatel může tuto funkci volat vícekrát, musíme vždy smazat obsah *ax*, jinak řečeno smažeme vykreslené sloupečky. To provedeme přes funkci *clear* na *ax*. Pro uložení vygenerovaných dat potřebujeme *list*, jenž má název *listhodnot*. Poté vytvoříme cyklus *for*, kterým do *listhodnot* vložíme náhodně vygenerovaná čísla od 20 do 100 (tolikrát kolik je prvků v *listu data*). Po dosazení hodnot do *listu* deklarujeme proměnné *nejvetsi* a *nejmensi*. Přes funkce *max* a *min* s argumentem *listhodnot* ukládáme maximální a minimální hodnoty do příslušných proměnných. Dále vytvoříme pro každý sloupeček barvu. Pro zkrácení zápisu využijeme ternárního operátoru. Jsou zde podmínky, kterými se ptáme, zda daný prvek v *listhodnot* je roven *nejmensi* (červená barva), *nejvetsi* (zelená barva). Pokud to není ani jedna z nich nastaví se modrá barva. Pro procházení prvků využíváme cyklu *for*.

```
barvy=['red' if x == nejmensi else 'green' if x == nejvetsi else 'blue' for x in listhodnot]
```

Obrázek 73 Ukázka ternárního operátoru (Zdroj: vlastní)

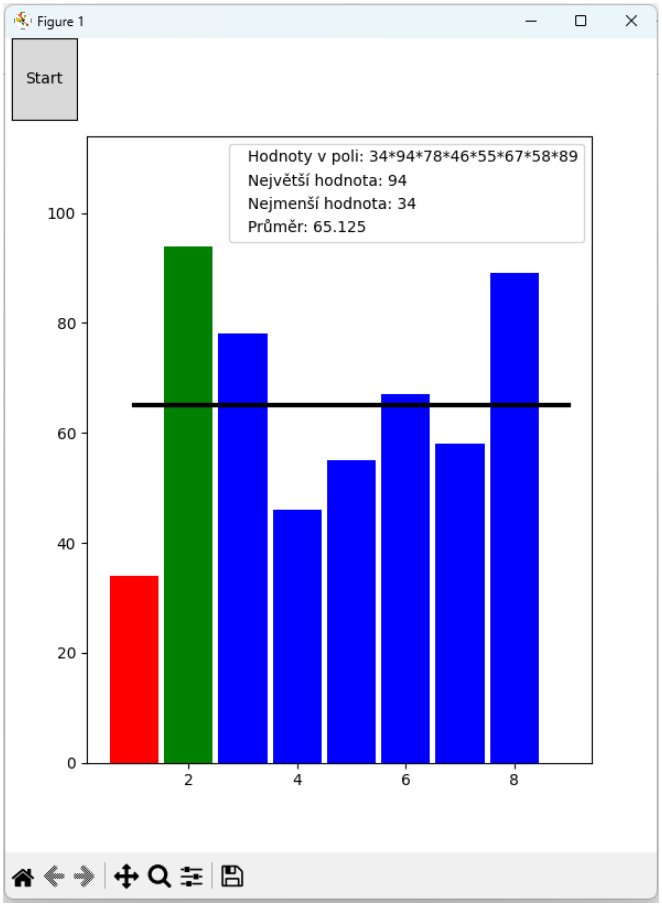
Dále vytvoříme další cyklus *for*, kterým vykreslíme jednotlivé sloupečky s barvami a vypočteme součet hodnot prvků v *listhodnot*, ten budeme potřebovat pro výpočet průměru. Pro průchod všech prvků *listu* deklarujeme proměnnou *x* a pro sčítání všech hodnot proměnnou *soucet*. V cyklu pro vykreslení sloupečku využíváme funkci *bar* volanou na *ax*. Obsahuje argumenty *data* (počet sloupců k vykreslení), *listhodnot* (výška sloupce), šířku a barvu sloupce. Po vykreslení sloupců vypočteme průměr jako podíl mezi *soucet* a délka *listhodnot*. Výsledek uložíme do proměnné *prumer*. Dále si deklarujeme proměnnou *vypis*, do níž jako v původní úloze vložíme vypsání jednotlivých prvků z *listu*,

mezi kterými je znak hvězdička. Nyní máme vytvořeny všechny podklady pro vytvoření legendy. Tu vytvoříme přes funkci *legend*, kdy do jejího parametru přes formátované řetězcové literály vkládáme nadpisy a hodnoty k příslušné legendě.

```
ax.legend([f'Hodnoty v poli: {vypis}',f'Největší hodnota: {nejvetsi}',
```

Obrázek 74 Ukázka vytvoření legendy (Zdroj: vlastní)

Pro zobrazení průměru využíváme linii, tu vykreslíme přes funkci *plot*, u které jako první parametr nastavujeme počáteční a konečný bod na ose x neboli délku. Druhým parametrem jsou výšky vykreslení, kterým je přiřazena hodnota proměnné *prumer*. Dále nastavíme barvu a šířku linie. Jako poslední ve funkci *vykreslit* je aktualizace *fig*. Pro spuštění funkce *vykreslit* musíme na proměnné *tlacitko* volat metodu *on_clicked*. Funkci *vykreslit* musíme vložit jako argument metodě *on_clicked*, která ji zavolá po kliknutí na tlačítko. Na konci dokumentu na *plt* voláme funkci *show*, kterou vykreslujeme hlavní okno.



Obrázek 75 Výsledek Grafy s externími moduly

ZÁVĚR

Bakalářská práce je primárně zaměřena na transformaci kódu z jazyku JavaScript do Pythonu. Teoretická část je rozdělena na „obecnou teorii“ týkající se např. vyššího jazyku a interpreta. V kapitolách JavaScript a Python nalezneme podkapitoly týkající se základní historie, syntaxe atd. U kapitoly JavaScript je i základní popis týkající se HTML a CSS. Je to z důvodu, že tyto dva jazyky se využívaly pro zobrazení JS kódu. Stěžejní částí této práce jsou vypracované samotné úlohy. Kód v JavaScriptu byl napsán na seminářích předmětu PGM1P, tudíž v praktické části nejsou úkoly tak podrobně popsány jako v jazyce Python. Ten je naopak popisován velice podrobně. Je to z důvodu toho, aby se čtenář seznámil s hlavními rozdíly mezi jazyky.

Mezi tyto rozdíly můžeme zahrnout samotné zobrazování výstupu kódu JavaScript i Python, kdy JS zobrazujeme přes webový prohlížeč. Úlohy Pythonu jsou zobrazovány přes modul zabudovaný v Pythonu s názvem Tkinter. O funkčnosti transformovaných úloh je možné prohlásit, že jsou hodně podobné nebo totožné. Je důležité zmínit, že vypracované úlohy do Pythonu se ukázaly někdy vhodné, ale pro některé úlohy bylo nutné vytvořit složitější zápisy. V praktické části u Pythonu se úlohy vypracovávaly i pomocí externích modulů. Úlohy s využitím modulů nebyly vypracovány všechny, a to z důvodu, že na práci s moduly byly příliš jednoduché. Řešení by se nezjednodušilo, ani nepřineslo zlepšení funkcionality. V další práci by bylo vhodné vypracovat úlohy, které by neodrážely pouze přesné převedení úloh z JS do Pythonu při zachování jejich funkcionality, ale byly i vhodné pro jeho výuku takovým způsobem, jako to momentálně dělají úlohy JS v předmětu PGM1P.

RESUMÉ

The main aim of this bachelos's thesis was to transform JavaScript tasks into Python. Even at first glance, it is safe to say that Python code is different from JavaScript. Since Python does not use braces but indentation. A big difference is also the display of the tasks themselves and that is when we use HTML at JS and CSS for graphic editing. The Tkinter module is used for Python tasks. Other differences can be find in the codes themselves, for example the task about saving cars with given values is solved in JS through objects. In Python on the other hand, through dictionaries as it was necessary to create helper function for gradual saving in a list. Another example is the "Dictation task" which has the function of replacing characters in a string. Those tasks had some similar parts, but the text replacement and coloring were different. Tasks that were simple in JS became comparatively more difficult in Python.

Differences also occurred in code processing with and without the use of external modules. For example, it is possible to use advanced functions such as standard deviation, when using external modules in mathematical operations. There was also a significant difference in the task related to drawing and moving the ball, Pygame uses other notations. If it is possible to summarize it simply, it is necessary to adapt notations for a specific module in order to achieve more advanced work with module.

To enable the individual tasks to be used in teaching, their assignments would have to be redesigned. It would be possible to focus on this in the follow-up work, where the assignment would be modified and the tasks could subsequently be used in teaching programming.

SEZNAM LITERATURY

1. **Turing.** Your Ultimate Guide To Visual Studio vs Visual Studio Code. *turing.* [Online] [Citace: 22. červen 2023.] https://www.turing.com/kb/ultimate-guide-visual-studio-vs-visual-studio-code.

2. **Visual Studio Code.** Extension Marketplace. *code.visualstudio.* [Online] 6. Srpen 2023. [Citace: 22. Červen 2023.] https://code.visualstudio.com/docs/editor/extension-marketplace.

3. **JetBrains.** PyCharm Community Edition vs PyCharm Pro. *jetbrains.* [Online] [Citace: 22. Červen 2023.] https://www.jetbrains.com/products/compare/?product=pycharm-ce&product=pycharm.

4. **JetBrains**. PyCharm Features. *jetbrains.* [Online] [Citace: 22. Červen 2023.] https://www.jetbrains.com/pycharm/features/.

5. **CodePen.** About CodePen. *codepen.* [Online] [Citace: 22. Červen 2023.] https://codepen.io/about/.

6. **Buckley, Ian.** 8 Awesome CodePen Features for Programming and Web Development. *makeuseof.* [Online] 26. Červenec 2018. [Citace: 22. Červen 2023.] https://www.makeuseof.com/tag/best-codepen-features/.

7. **Isaac Computer Science.** Low-level languages. *isaaccomputerscience.* [Online] [Citace: 22. Červen 2023.] https://isaaccomputerscience.org/concepts/sys_proglang_low_level.

8. **STRÁNKY K VÝUCE INFORMATIKY.** 14. Programovací jazyky. *ivt.mzf.* [Online] [Citace: 22. Červen 2023.] http://www.ivt.mzf.cz/seminar/14-programovaci-jazyky/.

9. **Isaac Computer Science.** High-level languages. *isaaccomputerscience.* [Online] [Citace: 22. Červen 2023.] https://isaaccomputerscience.org/concepts/sys_proglang_high_level.

10. **GeeksforGeeks.** Introduction to Interpreters. *geeksforgeeks.* [Online] 10. Červen 2023. [Citace: 22. Červen 2023.] https://www.geeksforgeeks.org/introduction-to-interpreters/.

11. **Programiz.** Interpreter Vs Compiler : Differences Between Interpreter and Compiler. *programiz.* [Online] [Citace: 22. Červen 2023.] https://www.programiz.com/article/difference-compiler-interpreter.

12. **Sheldon, Robert.** compiler. *techtarget.* [Online] Duben 2022. [Citace: 22. Červen 2023.] https://www.techtarget.com/whatis/definition/compiler.

13. **Kumar, Subodh.** Why JavaScript is called Interpreted or JIT(Just In Time) Compiled. *javascript.plainenglish.* [Online] 4. Leden 2022. [Citace: 22. Červen 2023.] https://javascript.plainenglish.io/why-javascript-is-called-interpreted-or-jit-just-in-time-compiled-c8cc490682bd.

14. **Doglio, Fernando.** The JIT in JavaScript: Just In Time Compiler. *blog.bitsrc.* [Online] 12. Srpen 2020. [Citace: 22. Červen 2023.] https://blog.bitsrc.io/the-jit-in-javascript-just-in-time-compiler-798b66e44143.

15. **W3Schools.** What is JavaScript? *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/whatis/whatis_js.asp.

16. **W3Schools**. JavaScript History. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/js/js_history.asp.

17. **Javiya, Rushi.** A Brief History of JavaScript? *tutorialspoint.* [Online] 28. Prosinec 2022. [Citace: 22. Červen 2023.] https://www.tutorialspoint.com/a-brief-history-of-javascript.

18. **Máca, Jindřich.** Lekce 1 - Úvod do Node.js. *itnetwork.* [Online] [Citace: 22. Červen 2023.] https://www.itnetwork.cz/javascript/nodejs/uvod-do-nodejs.

19. **MDN Web Docs .** WebAssembly. *developer.mozilla.* [Online] 31. Květen 2023. [Citace: 22. Červen 2023.] https://developer.mozilla.org/en-US/docs/WebAssembly.

20. **Node.js.** The V8 JavaScript Engine. *nodejs.* [Online] [Citace: 22. Červen 2023.] https://nodejs.dev/en/learn/the-v8-javascript-engine/.

21. **V8 JavaScript engine.** What is V8? *v8.* [Online] [Citace: 22. Červen 2023.] https://v8.dev/.

22. **W3Schools.** JavaScript Syntax. *w3schools.* [Online] [Citace: 24. Červen 2023.] https://www.w3schools.com/js/js_syntax.asp.

23. **W3Schools**. JavaScript Data Types. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/js/js_datatypes.asp.

24. **TutorialsTeacher.** Difference between null and undefined in JavaScript. *tutorialsteacher.* [Online] [Citace: 22. Červen 2023.] https://www.tutorialsteacher.com/javascript/javascript-null-and-undefined.

25. **W3Schools.** JavaScript Variables. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/js/js_variables.asp.

26. **Zlatkov, Alexandr.** How JavaScript works: memory management + how to handle 4 common memory leaks. *medium.* [Online] 13. Září 2017. [Citace: 22. Červen 2023.] https://medium.com/sessionstack-blog/how-javascript-works-memory-management-how-to-handle-4-common-memory-leaks-3f28b94cfbec.

27. **W3Schools.** JavaScript Functions. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/js/js_functions.asp.

28. **W3Schools**. JavaScript Operators. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/js/js_operators.asp.

29. **W3Schools**. JavaScript if, else, and else if. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/js/js_if_else.asp.

30. **W3Schools**. JavaScript Switch Statement. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/js/js_switch.asp.

31. **W3Schools**. JavaScript For Loop. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/js/js_loop_for.asp.

32. **W3Schools**. JavaScript While Loop. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/js/js_loop_while.asp.

33. **W3Schools**. JavaScript Arrays. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/js/js_arrays.asp.

34. **vavyskov.** Značkovací jazyk (HTML). *web.vavyskov.* [Online] [Citace: 22. Červen 2023.] https://web.vavyskov.cz/znackovaci-jazyk.html.

35. **Hél, Samuel.** Lekce 1 - Základní struktura HTML. *itnetwork.* [Online] [Citace: 22. Červen 2023.] https://www.itnetwork.cz/html-css/html5/zakladni-struktura-html.

36. **W3Schools.** HTML Introduction. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/html/html_intro.asp.

37. **Vorbová, Renáta.** Co je nutné vědět o CSS – 1. díl. *blog.shoptet.* [Online] 23. Září 2009. [Citace: 22. Červen 2023.] https://blog.shoptet.cz/co-je-nutne-vedet-o-css-1-dil/.

38. **W3Schools.** CSS Selectors. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/css/css_selectors.asp.

39. **W3Schools**. w3schools. *How To Add CSS.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/css/css_howto.asp.

40. **Čápka, David.** Lekce 1 - Úvod do CSS frameworku Bootstrap. *itnetwork.* [Online] [Citace: 22. Červen 2023.] https://www.itnetwork.cz/html-css/bootstrap/kurz/uvod-do-css-frameworku-bootstrap.

41. **python.** What is Python? Executive Summary. *python.* [Online] [Citace: 22. Červen 2023.] https://www.python.org/doc/essays/blurb/.

42. **EXYTE.** Stručná historie Pythonu. *exyte.* [Online] 3. Listopad 2020. [Citace: 22. Červen 2023.] https://exyte.com/blog/a-brief-history-of-python.

43. **SohomPramanick.** History of Python. *geeksforgeeks.* [Online] 26. Listopad 2022. [Citace: 22. Červen 2023.] https://www.geeksforgeeks.org/history-of-python/.

44. **DataCamp.** Python 2 vs 3: Everything You Need to Know. *datacamp.* [Online] Srpen 2022. [Citace: 22. Červen 2023.] https://www.datacamp.com/blog/python-2-vs-3-everything-you-need-to-know.

45. **W3Schools.** Python Syntax. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/python/python_syntax.asp.

46. **W3Schools**. Python Data Types. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/python/python_datatypes.asp.

47. **Kareem, Hafeezul.** Is Python Dynamically Typed Language? *tutorialspoint.* [Online] 30. Červenec 2019. [Citace: 22. Červen 2023.] https://www.tutorialspoint.com/is-python-dynamically-typed-language.

48. **W3Schools.** Python Variables. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/python/python_variables.asp.

49. **W3Schools**. Python - Global Variables. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/python/python_variables_global.asp.

50. **W3Schools**. Python Functions. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/python/python_functions.asp.

51. **W3Schools**. Python Operators. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/python/python_operators.asp.

52. **W3Schools**. Python If ... Else. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/python/python_conditions.asp.

53. **W3Schools**. Python While Loops. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/python/python_while_loops.asp.

54. **W3Schools**. Python For Loops. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/python/python_for_loops.asp.

55. **Programiz.** Python List. *programiz.* [Online] [Citace: 22. Červen 2023.] https://www.programiz.com/python-programming/list.

56. **W3Schools.** Python Dictionaries. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/python/python_dictionaries.asp.

57. **W3Schools**. Python PIP. *w3schools.* [Online] [Citace: 22. Červen 2023.] https://www.w3schools.com/python/python_pip.asp.

58. **Turing.** Creating and Publishing Python Packages to PyPI. *turing.* [Online] [Citace: 22. Červen 2023.] https://www.turing.com/kb/how-to-create-pypi-packages.

59. **Sanwo, Stephen.** How to Set Up a Virtual Environment in Python – And Why It's Useful. *freecodecamp.* [Online] 11. Duben 2022. [Citace: 22. Červen 2023.] https://www.freecodecamp.org/news/how-to-setup-virtual-environments-in-python/.

SEZNAM OBRÁZKŮ, TABULEK, GRAFŮ A DIAGRAMŮ

PŘÍLOHY

code_bc_javascript

code_bc_modules_python

code_bc_nomodules_python