Skriptovací programovací jazyky a jejich aplikace

ZS 2020/2021

Informace

http://mrl.cs.vsb.cz/people/holusa/spja_course

- Bodování:
  - Úlohy na cvičení 6-30 bodů
  - Projekt 10-30 bodů
  - Závěrečný test z programování 15-40 bodů

Informace

- Co vše proberem?
  - Základy jazyka (syntaxe, funkce, kolekce)
  - OOP
  - Práce se soubory
  - XML, XML-RPC
  - Django Framework
  - AI, GUI

Skriptovací jazyky

- Jazyky, které obvykle slouží k manipulaci s jinými programy nebo systémy nebo automatizaci úloh
- Typické vlastnosti:
  - Interpretované jazyky
  - Dynamické typování

Interpretované jazyky

- Kompilované jazyky: edit – compile – link – run
  - Po každé úpravě kódu je nutný překlad (může trvat dlouho)
- Interpretované jazyky: edit – run
  - Jednodušší ladění programu, dobrá přenositelnost

Dynamické typování

- Statické typování – datovým typem řeknene, co smí být do proměnné uloženeo

```
int a = 5; String s = 'Hello'; a = 'Hi';
```

- Dynamické typování – datový typ je pouze vlastnost, kontrola až za chodu programu

```
a = 5; s = 'Hello'; a = 'Hi';
```

Skriptovací jazyky

- Ruby
  - Framework Ruby on Rails
- JavaScript
  - Webové stránky
- PHP
  - Dynamické webové stránky
- **Python**

Python

- Vznikl v roce 1991, aktuální verze 3.8
- Aktuálně nejpoužívanější jazyk v programování umělé inteligence, ve statistice, ve psaní systémových testů
- Jednoduchá syntaxe
- Dynamické typování, automatická správa paměti
- Objektově orientovaný, imperativní, funkcionální jazyk
- Každá proměnná je odkaz na objekt

Python

- V čem psát zdrojový kód?
  - V jakémkoliv editoru textu – soubor stačí uložit s příponou .py (např. script.py)
  - Pro vyzkoušení si jednoduchých příkazů jazyka stačí interaktivní shell (ipython)
- Jak spustit skript?
  - Linux: `python3 script.py`
  - Windows: `py script.py`

Zápis čísel

- int, float, complex
- V Pythonu 2 existoval také typ long, nyní jsou jakkoliv dlouhá čísla typu int
- Desetinná čísla jsou omezena na 17 míst
  - Způsoby zápisu

```
0.4564513215645132105647 85
0.432_232_432_111  #pouze pro přehlednost programátora
123e-3
123e-30
```

Zápis čísel

- Complex
  - Reálná a imaginární část

```
In : 3+1j
Out: (3+1j)
In : 3+1j + 2+3j
Out: (5+4j)
```

Zápis čísel

- Operátory

  sčítání (+) a odčítání (-)

  násobení (*), dělení (/), celočíselné dělení (//), zbytek po dělení (%)

  umocňování (**)

  bitové operace (~ & | << >>)

Pravdivostní výrazy

- True, False
- `>` `<` `>=` `<=` `==` `!=` `is` `is not`

hodnoty | objekty

Řetězce

```
s1 = "Hello"      s2 = 'Hello'

"Lze i 'kombinovat', jen je treba 'ukoncit'
retezec stejnym znakem"
```

- Lze použít i některé operátory

```
'h'+'i'      #'hi'

'h'+2*'i'    #'hii'
```

Řetězce

Přístup ke znakům/podřetězce

```
s = 'Hello'
s[0] = 'H'
s[:2] = 'He'
s[2:] = 'llo'
```

Proměnné

```
a = 5              type(a)   # int
a = 5.6            type(a)   # float
s = "hello"        type(s)   # str
type(a) == float             # True
isinstance(a, float)         # True
a = None  # objekt který nic neobsahuje
```

Proměnné

Proměnné obsahují odkazy na objekty

```
a = "hello "   b = "world"

ab = a+b

ab2 = ab

ab == a+b   #True

ab is a+b   #False

ab2 is ab   #True
```

Proměnné

A taky….

```
a = 256   b = 256

c = 257   d = 257

a == b # True          a is b # True

c == d # True          c is d # False
```

Funkce

Užitečné funkce:

float, int, str - přetypování

id – id objektu

type – datový typ objektu

max, min – extrémy v sekvenci prvků

len – délka

range(start, stop) – posloupnost čísel od start do stop

Syntaxe jazyka

Funkce

```
def <name>(<params>): <function-body>


def soucet(a,b):
   return a+b


print(soucet(4,3))
print(soucet("he","llo"))
```

Syntaxe jazyka

Komentáře

```
# jednořádkový
""" víceřádkový
komentář """
```

Syntaxe jazyka

C++:

```
if ( a > 0 ) {
   printf("Positive\n");
}
else if (a < 0 ) {
   printf("Negative\n");
}
else {
   printf("Zero\n");
}
```

Python:

```
if a > 0:
   print("Positive")
elif a < 0:
   print("Negative");
else:
   print("Zero");
```

Syntaxe jazyka

C++:

```
for (int i=0; i<10; i++)
{
  printf("Number: %d\n", i);
  printf("Square: %d\n", i*i);
}
```

Python:

```
for i in range(10):
    print("Number", i)
    print("Square", i ** 2)
```

Syntaxe jazyka

Důležité je odsazení

```
for i in range(10):
    print("Number", i)
    if i % 2:
        print("Odd")
    else:
        print("Even")
```

Syntaxe jazyka

Podmínky

```
if <expr>: <nested block>
0+ elif <expr>: <nested block>
optionally: else: <nested block>
```

Cykly

```
while <expr>: <nested block>
for <name> in <iterable>:
   break, continue
```

Formátování textu

- Operátor %

```
formát % hodnoty

"%s obsahuje %d členů s průměrem %.1f" %
("Skupina", 5, 32.1234)
   "Skupina 5 32.1"
```

Formátování textu

- Funkce format()

```
"{} {} {}".format("Skupina", 5, 32.1234)
  "Skupina 5 32.1234"

"{1} obsahuje {0} členů s průměrem
{2}".format("Skupina", 5, 32.1234)
  "5 Skupina 32.12"

"{0}: {1:.2f}".format(5, 32.1234)
  "5 32.12"
```

Formátování textu

- Funkce format()

```
"{} {} {}".format("Skupina", 5, 32.1234)
  "Skupina 5 32.1234"

"{1} obsahuje {0} členů s průměrem
{2}".format("Skupina", 5, 32.1234)
  "5 Skupina 32.12"

"{0}: {1:.2f}".format(5, 32.1234)
  "5 32.12"
```

Kontejnery

- Datové struktury pro uchovávání skupin dat
- 4 druhy:
  - seznamy
  - n-tice
  - množiny
  - slovníky

Seznam (list)

- Proměnný objekt

```
lst = [1, ‘a’, [2, True]]
lst[0] = 2
```

- Je možné provádět operace sčítání a násobení konstantou
- List comprehension

```
[x**2 for x in lst if x > 2]
```

- Délka seznamu `len()`, suma hodnot `sum()`
- Je prvek v listu? `if x in lst`

Seznam (list)

- Přidávání prvků na konec seznamu

  `append(), extend()`

- Užitečné metody

  `index(), insert(), remove(), reverse(), sort()`

- Vícerozměrné seznamy

  ```
  lst=[[1,2,3], [4,5,6], [7,8,9]]
  lst[row][col]
  ```

N-tice (tuple)

- Nemodifikovatelná posloupnost prvků

```
tup = (1, ‘a’, [2, True])
tup[0] = 2 #nelze
```

- Je možné provádět operace sčítání a násobení konstantou
- Hvězdičkové pravidlo

```
a, b, *c = (1,2,3,4,5,6)
```

Množina (set)

- Neuspořádaná kolekce jedinečných hodnot

```
s = {1, 2, ‘a’, (3, 4)} #pouze neměnné objekty
```

- `union (|), intersection (&), difference (-)`
- Užitečné metody

```
add(), remove(), issubset()
```

Slovník (dict)

- Uchovává uspořádané dvojice klíč, hodnota

```
d = {'a':1, 'b':2}
d['a'], d['c'] = 3
d.keys(), d.values(), d.items()
```

- Dict comprehension

```
{c:ord(c) for c in "python"}
```

Definice funkcí

- Parametry jsou předávány hodnotou
- Implicitní hodnoty argumentů

```
def kruh(x, y, rad=5)
```

- Pojmenované argumenty

```
kruh(y=2, x=4)
```

Definice funkcí

- Argument s hvězdičkou

```
def fun(a=2, *b)
fun()
fun(1)
fun(1,2)
fun(1,2,3)
```

Definice funkcí

- Dvouhvězdičkový parametr

```
def fun(a=2, **b)
fun()
fun(1)
fun(1,b=2)
fun(1,b=2,c=3)
```

Moduly

- Importování modulů

```
import math                         math.sqrt(9)
from math import sqrt               sqrt(9)
import math as m                    m.sqrt(9)
from math import sqrt as odm        odm(9)
```

- Vlastní moduly

Práce se soubory

- Moduly pro práci se soubory
  - `os, os.path`
    - Seznam souborů v adresáři

      `os.listdir('dir')`
    - Vytvoření adresáře

      `os.mkdir(path)`
    - Existuje soubor/adresář?

      `os.path.exists(path)`

      `os.path.isdir(path), os.path.isfile(path)`

Práce se soubory

- Otevírání souborů

```
open(file, mode='r') #mode: r, w, a, b
```

- Zápis

```
f = open('file.txt', 'w')
f.write("Hello\n")
print("Hi", file=f)
f.close()
```

Práce se soubory

- Otevírání souborů

```
open(file, mode='r') #mode: r, w, a, b
```

- Čtení

```
f = open('file.txt', 'r')
lines = f.readlines()
for line in f:
   #process line
   line=line.rstrip() # odstrani \n na konci
f.close()
```

Práce se soubory

- Otevírání souborů

```
open(file, mode='r') #mode: r, w, a, b
```

- Alternativní způsob otevření

```
with open('file.txt', 'r') as f:
    for line in f:
        line=line.rstrip()
            print(line)
```

- Není nutné zavírat

Práce se soubory

- Binární soubory

```
open(file, mode="wb")
byte_arr=[5, 12, 8]
binary_format = bytearray(byte_arr)
f.write(binary_format)

fr = open("test.bin", "rb")
data = fr.read()
```

Práce se soubory

- Binární soubory

```
arr=[2.34, 1.8, 4.121]
a = array('d', arr)
# 'd' – double, 'f' – float, 'q' – long long
a.tofile(f)

arr = array('d')
arr.frombytes(fr.read())
```

Výjimky

- Objekt vytvořený v okamžiku, kdy nastane výjimečná situace
- Python využívá `try` a `except` bloky k ošetření výjimek
- `raise` k vytvoření výjimky

Výjimky

```
try :
   #nějaký kód, který vyvolá výjimku
   ...
except <exception>:
   #kód, ošetřující výjimku
finally :
   #kód vykonaný na závěr
```

Výjimky

- Vlastní výjimka

```
class MyError(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return "Chyba "+self.value

raise MyError("0")
```

OOP

- Základní princip
  - Programy jsou simulací skutečného nebo námi vymyšleného virtuálního světa (světy objektů, které spolu komunikují)
  - Třídy, instance třídy
  - Dědičnost, zapouzdření, polymorfismus

OOP

```
class <name >(<bases>):
<body>
```

- v `<body>` definujeme metody s použitím `def`
- Pokud třída nedědí, použije se `object`
- Třídní vs instanční proměnné

OOP

```
class Document(object):
    no_of_docs = 0    # tridni promenna

    def __init__(self, content): # inicializátor třídy
        self.__content = content  # instancni promenna
        Document.no_of_docs += 1

    def get_content(self):      # metoda
        return self.__content
d = Document("Text z dokumentu 1")
```

OOP

- Statické metody `@staticmethod`
  - nezajistí, že bude metoda statická, ale že ji jako statickou bude možno volat na instanci.
  - nemá jako svůj první argument proměnnou self (nemůže být volána na objektu)
- Vlastnosti (properties)

```
content = property(get_content, set_content)
```

OOP

- Dědičnost
  - Kompozice vs dědičnost
    - Je X součástí Y?
    - Je X speciální případ Y?

```
class SomeClass(ParentClass):
    def __init__(self, paramForParent,
    paramForThisClass):
        super().__init__(paramForParent)
        ...
```

OOP

- Přetížení operátorů (např. `__lt__`, `__gt__`, `__add__`, `__sub__`)

```
class Vector:
    def __init__(self, *vec):
        self.__vec = vec

    def __add__(self, other):
        vec = []
        if len(self.__vec) == len(other.__vec):
            for no, item in enumerate(self.__vec):
                vec.append(item + other.__vec[no])
        return vec
```

XML

- Extensible Markup Language
  - Značkovací jazyk určen především pro výměnu dat mezi aplikacemi
  - Řídí se jasně danými pravidly pro strukturu dokumentu
    - Má jeden kořenový (root) element
    - Obsahuje elementy, které musí být ohraničeny startovací a ukončovací značkou (`<`, `>`, `/>`)
    - Všechny hodnoty atributů musí být uzavřeny v uvozovkách
    - Elementy mohou být vnořeny, ale nemohou se překrývat

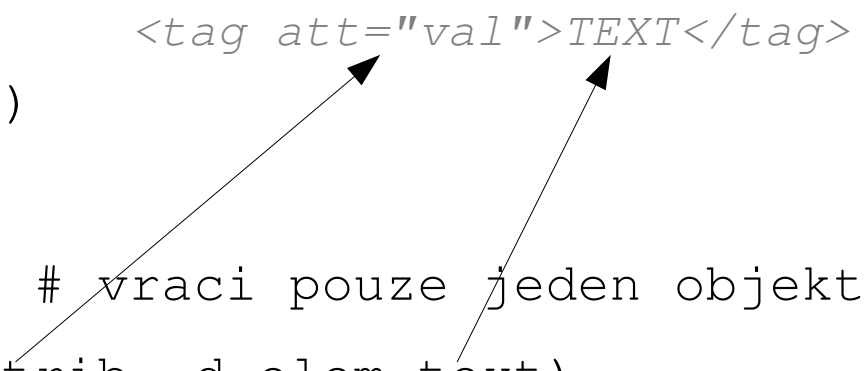
https://cs.wikipedia.org/wiki/Extensible_Markup_Language

XML

```
<?xml version="1.0" encoding="UTF - 8" ?>
<menza> #root element
<tyden>35</tyden> #element s textem
<datum den="Pondeli"> #element s atributem den
<jidlo nazev="Bramborove placky"> #element s atributem nazev
<ingredience nazev="brambory" /> #vnoreny element (hned ukončen)
<ingredience nazev="mouka" />
<ingredience nazev="vejce" />
</jidlo> #ukonceni elementu jidlo
</menza> #ukonceni root elementu
```

XML

```
import xml.etree.ElementTree as ET


tree = ET.parse("menza.xml")
root = tree.getroot()
d_elem = root.find('datum') # vraci pouze jeden objekt
print(d_elem.tag, d_elem.attrib, d_elem.text)
print(d_attrib['den'])
datum_elem = root.findall('datum') # vraci seznam objektu
datum_elem.findall('jidlo') # postupne se zanorujeme dale
```



XML

- Vytvoření XML

```
root = ET.Element("restaurace_menu")
week = ET.SubElement(root, "tyden")
week.text='35'
day = ET.SubElement(root, "datum")
day.set("den", "Pondeli")
jidlo = ET.SubElement(day, "jidlo")
jidlo.set("nazev", "Smazak")
tree = ET.ElementTree(root)
tree.write("menu.xml")
```

```
<restaurace_menu>
 <tyden>35</tyden>
 <datum den="Pondeli">
  <jidlo nazev="Smazak" />
 </datum>
</restaurace_menu>
```

HTTP

- Protokol určený pro komunikaci
- Klient pošle požadavek na server, server odpoví
- Metody:
  - GET – poažadavek na zobrazení/získání dat
  - POST – odeslání dat na server
  - PUT, HEAD, DELETE, ...

Vzdálené volání procedur

- Remote Procedure Call
- Komunikace založená na HTTP
- Klient volá metody na serveru
  - Klient vytvoří “balíček” obsahující volanou metodu a její parametry
  - Tento “balíček” je přenesen na server
  - Na serveru proběhne rozbalení a provedení procedury
  - Server vytvoří “balíček” s výsledkem a pošle zpět klientovi
  - Klient data rozbalí a nějak zpracuje (např. zobrazí)

XML-RPC

- Jaký je formát “balíčků” přenášených mezi klientem a serverem?
- Jednou z možností je použít XML formát – data jsou přenášena v tomto formátu

- Představme si, že je existuje serverová aplikace poskytující informace o státech. Chceme např. zjistit hlavní město země, pro tohle na serveru existuje metoda `getCapital(str)`

XML-RPC

- Jak vypadají přenášená data? (hlavička + tělo)
- Tělo požadavku:

```
<?xml version="1.0" ?>
<methodCall>
    <methodName>getCapital</methodName>
    <params>
        <param>
            <value><string>Czechia</string></value>
        </param>
    </params>
</methodCall>
```

XML-RPC

- Jak vypadají přenášená data? (hlavička + tělo)
- Tělo odpovědi:

```
<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><string>Prague</string></value>
      </param>
   </params>
</methodResponse>
```

XML-RPC

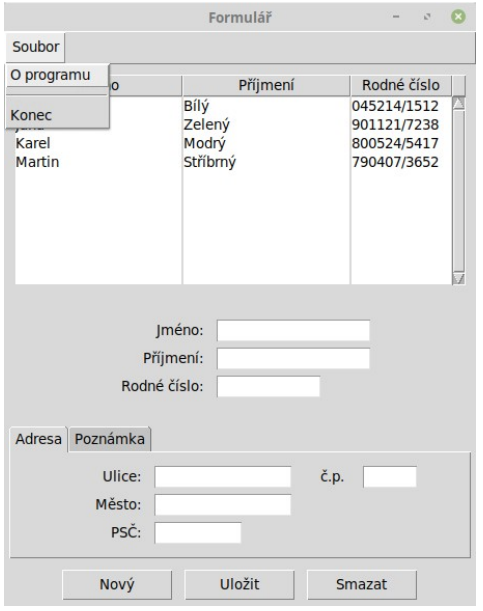
- Ačkoliv víme, jak pracovat s XML soubory v Pythonu, nemusíme to v práci s XML-RPC řešit, vyřeší to použité knihovny
- Použití je jednoduché, viz videa, kde si vytvoříme jednoduchý kalkulačkový server a z klienta budeme volat funkce

GUI

- Prozatím jsme vytvářeli aplikace s výstupem na konzoli
- Ukážeme, jak se dá v Pythonu vytvářet grafické rozhraní
- Existuje více knihoven pro tvorbu GUI, představíme si Tkinter

  https://docs.python.org/3/library/tkinter.html

  ```
  sudo apt install python3-tk
  ```



GUI

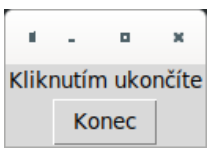
- GUI se tvoří tak, že používáme předem připravené prvky (tlačítka, popisky, zaškrtávací políčka atd.), které umisťujeme do okna s využitím vybraného správce rozložení
- Co vše lze použít a nastavit? https://www.tcl.tk/man/tcl8.6/TkCmd/contents.htm
- První program – vytvoříme okno s tlačítkem po jehož stisknutí se aplikace zavře

```
from tkinter import *
root = Tk() # hlavni okno aplikace
# vytvorime tlacitko v root - tedy v hlavnim okne
btn = Button(root, text="Konec", command=root.quit)
btn.pack() # vykreslime tlaciko
root.mainloop() # spustime beh aplikace
```

GUI

- Je také vhodné pro grafické okno vytvořit vlastní třídu

```
class MyGUI:
    def __init__(self, root):
        # pridame i popisek
        self.lbl = Label(root, text="Kliknutím ukončíte")
        self.btn = Button(root, text="Konec", command=root.quit)
        self.lbl.pack()
        self.btn.pack()

root = Tk()
app = MyGUI(root)
root.mainloop()
```



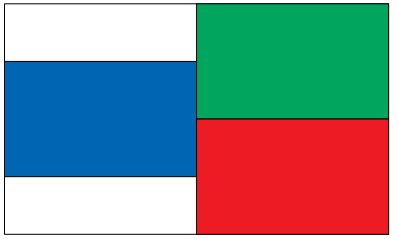
GUI – rozložení pack

```
obj.pack (side=?, fill=?, anchor=?)
```

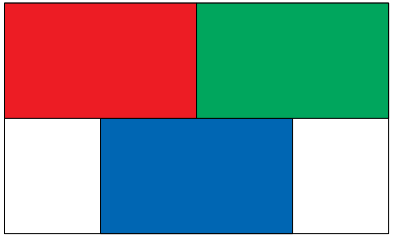
- `side` – strana, na kterou se komponenta umístí
  `(left, right, top, bottom)`

  záleží na pořadí

```
self.blue.pack(side="left")
self.red.pack(side="bottom")
self.green.pack()
```



```
self.blue.pack(side="bottom")
self.red.pack(side="left")
self.green.pack()
```



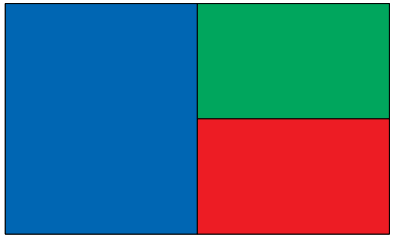
GUI – rozložení pack

```
obj.pack (side=?, fill=?, anchor=?)
```

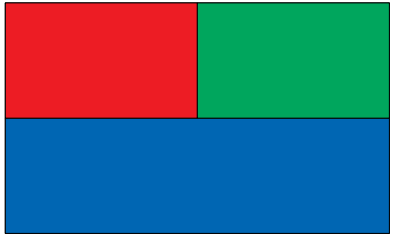
- `fill` – vyplní volnou plochu alokovanou pro komponentu

  `X, Y, BOTH`

```
self.blue.pack(side="left", fill=Y)
self.red.pack(side="bottom")
self.green.pack()
```



```
self.blue.pack(side="bottom", fill=X)
self.red.pack(side="left")
self.green.pack()
```



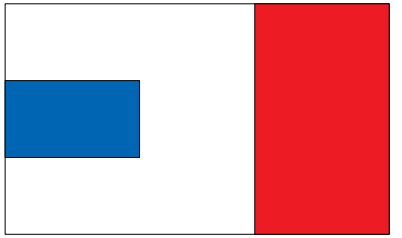
GUI – rozložení pack

```
obj.pack (side=?, fill=?, anchor=?)
```

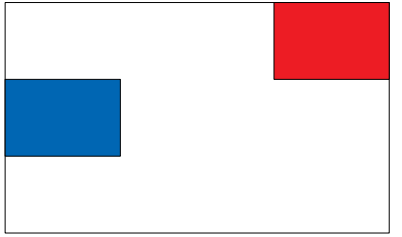
- `anchor` – ukotvení na stranu

  `'n', 'w', 'e', 's'`

```
self.blue.pack(side="left")
self.red.pack(side="bottom", fill=Y,
              expand=1, anchor="e")
```



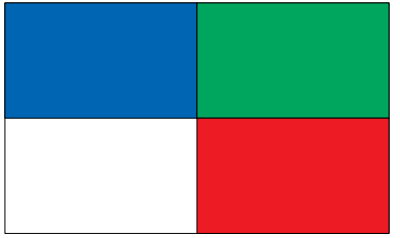
```
self.blue.pack(side="left")
self.red.pack(side="bottom", expand=1,
              anchor="ne")
```



GUI - rozložení grid

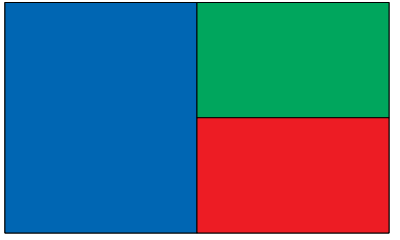
- `row, column` – index v mřížce

```
self.blue.grid(row=0, column=0)
self.red.pack(row=0, column=1)
self.green.pack(row=1, column=1)
```



- `rowspan, columnspan` – kolik buněk komponenta zaujme

```
self.blue.grid(row=0, column=0, rowspan=2)
self.red.pack(row=0, column=1)
self.green.pack(row=1, column=1)
```

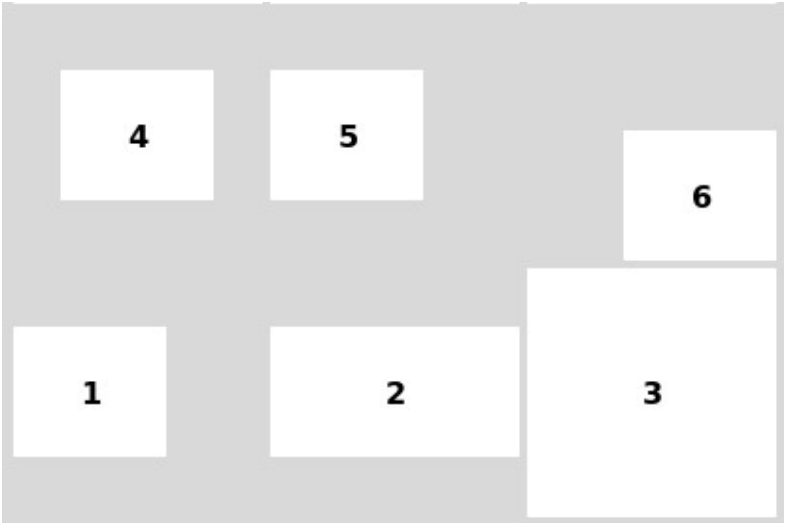


GUI - rozložení grid

- `sticky` – rozšiřování v rámci buňky

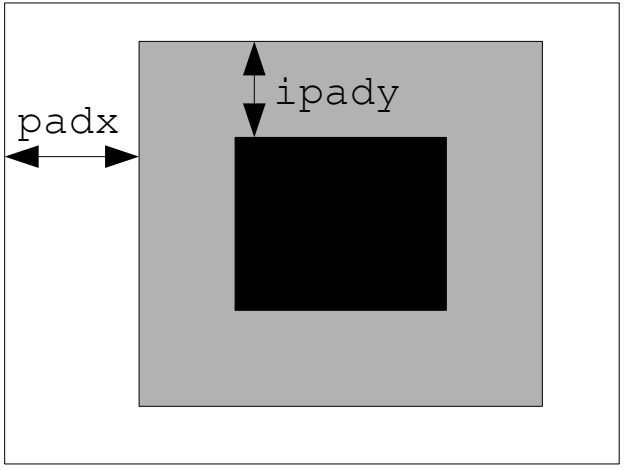
`N,W,E,S`

```
b1.grid(row=1,column=0,sticky=W)
b2.grid(row=1,column=1,sticky=W+E)
b3.grid(row=1,column=2,sticky=W+E+N+S)
b4.grid(row=0,column=0)
b5.grid(row=0,column=1,sticky=W)
b6.grid(row=0,column=2,sticky=E+S)
```



GUI – mezery na okraji

- `padx, pady` – vnější okraje
- `ipadx, ipady` – vnitřní okraje



GUI - Nastavení vlastností

`root.title('App Name')` – popisek okna

`root.resizable(False, False)` – povolení změny velikosti okna ve směru x a y

`root.bind('<Return>', function)` – mapování událostí (klávesnice, myš)

```
def_font = tkinter.font.nametofont("TkDefaultFont")
def_font.config(size=16)
```
– změna defaultního fontu

GUI - Obrázek pomocí Canvas

```
ca = Canvas(frame, width=300, height=400)
photo = PhotoImage(file="img.png")
ca.create_image(150, 200, image=photo)
```
– pozice obrázku

Lze vykreslit i jiné prvky (úsečky, čtverce)

```
r = ca.create_rectangle(146, 292, 152, 80, fill="blue")
```

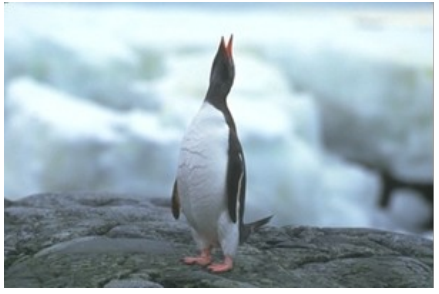
a lze je editovat

```
ca.coords(r, 146, 292, 152, 100)
```

Úvod do analýzy obrazu

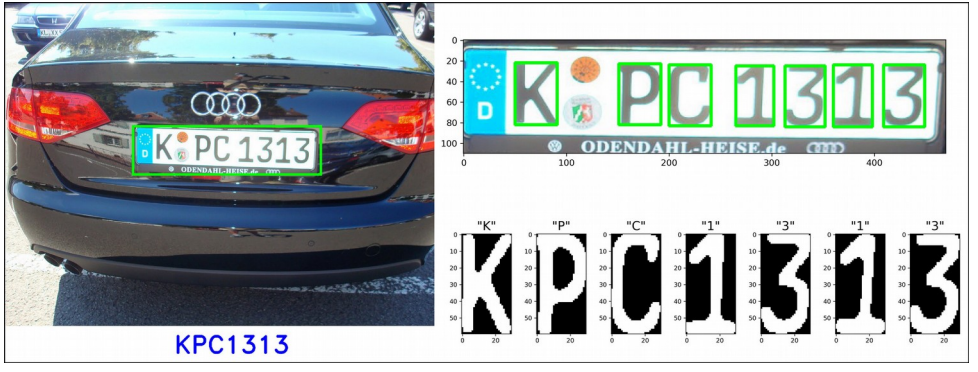
- Co lze např. dělat s obrazy?

Segmentace obrazu



Detekce obličejů

Čtení SPZ



Úvod do analýzy obrazu

- Existuje mnoho knihoven pro práci s obrazy
- Ukážeme si, jak pracovat s knihovnou OpenCV, kterou lze považovat za nejrozšířenější: www.opencv.org
- Knihovnu je třeba nainstalovat
- Ukážeme si, jak v Pythonu vytvořit virtuální prostředí (VE), ve kterém můžeme instalovat jakékoliv knihovny, aniž bychom je nainstalovali přímo do systému (můžeme tak mít více VE s různými verzemi knihoven)

Virtuální prostředí

- Vytvoříme prostředí

```
user@os: python3 -m venv nazev_prostredi
```

- Aktivujeme prostředí

```
user@os: . nazev_prostredi/bin/activate
```

- Toto se projeví tak, že název prostředí bude na začátku příkazového řádku. Nyní můžeme instalovat knihovny a spouštět skripty v rámci tohoto VE
- `(nazev_prostredi) user@os: python3 run.py`

Instalace OpenCV

- Instalovat lze přímo z balíčku pro OS, ukážeme si ale instalací pomocí `pip` – balíčkový systém pro Python
- Ve virtuálním prostředí spusťte

  ```
  pip install opencv-python==4.2.0.32 -vvv
  ```

  za == lze zadat verzi aplikace (pro aktuální verzi se část od == vynechá, ale v případě OpenCV mi instalace aktuální verze trvala cca 30 minut, takže jsem zvolil starší, která se nainstalovala rychleji, klidně ale použijte aktuální)

  `-vvv` zobrazí průběh instalace

- Pokud nemáte pip nainstalován:

  ```
  sudo apt install python3-venv python3-pip (Debian/Ubuntu)
  ```

Úvod do OpenCV

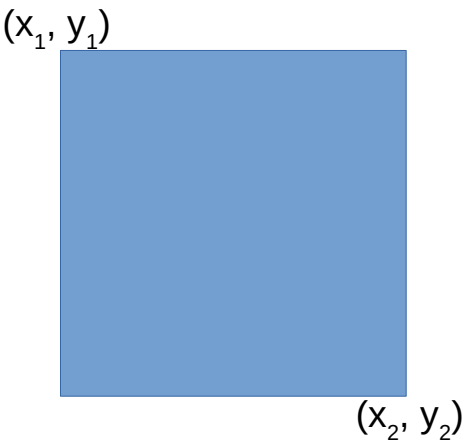
```
import cv2


# načtení obrázku (pokud se obrázek nenalezne, je img = None)
img = cv2.imread('img.jpg', cv2.IMREAD_COLOR)
# zobrazení a počkání na stisk klávesy
cv2.imshow('image', img)
cv2.waitKey(0)
# uložení obrázku na disk
cv2.imwrite('my_img.png', img)
```

Úvod do OpenCV

```
#převod barevného obrazu na obraz ve stupních šedi

img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# rozměry obrazu

img.shape  → (height, width, nchannels)

# přístup na pixely, vrací [B,G,R] v případě barevného obrazu
nebo hodnotu jasu v případě obrazu ve stupních šedi

pixel_value = img[row,column]

# změna hodnoty pixelu

img[row, column] = pixel_value
```

Úvod do OpenCV

```
#vykreslení útvarů do obrazu
cv2.line(img,(x1,y1),(x2,y2),(B,G,R), width)
cv2.rectangle(img,(x1,y1),(x2,y2),(B,G,R), width)
cv2.circle(img,(x,y), radius, (B,G,R), width)
```



Úvod do OpenCV

- Ukázka některých funkcí

```
# detekce hran

img_edges = cv2.Canny(img, threshold1, threshold2)

# detekce kružnic

circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,
scale_resolution, min_dist_between_circles)

# detekce obličejů

face_cascade = cv2.CascadeClassifier(cesta_ke_klasifikatoru)

faces = face_cascade.detectMultiScale(img)
```

Django

- Webový aplikační framework napsaný v Pythonu
- https://www.djangoproject.com/
- MTV architektura (Model, Template, View)
  - Model: objektově-relační mapování (modely = třídy v Pythonu převedeny do relační databáze)
  - Template: zobrazení dat v podobě webových stránek
  - View: funkce v Pythonu se vstupem (request) a výstupem (response)

Django

- Vytvoříme si webovou aplikaci, která bude spravovat data o fast-foodových restauracích

- Krok1: instalace Djanga: `pip install django` (opět možné ve virt. prostředí)
- Krok 2: vytvoříme nový projekt v Djangu

```
django-admin startproject restaurants
```

název projektu

Django

- Nyní máme adresář `restaurants` obsahující několik souborů

  `manage.py` – tímto skriptem budeme “ovládat” aplikaci (spouštět server, provádět operace s databází, atd.)

  `urls.py` – v tomto souboru budeme definovat webové adresy, které budou v rámci projektu dostupné

  `settings.py` – nastavení projektu (nastavení db, registrace aplikací v rámci projektu, atd.)

Django

- Spustíme server `python3 manage.py runserver` a otevřeme v prohlížeči na localhostu
- Projekt je souborem aplikací – může jich obsahovat více
- Vytvoříme tedy aplikaci s názvem `fastfoods`

  `python3 manage.py startapp fastfoods`
- Aplikaci je třeba přidat do seznamu v `settings.py`
- Vytvořil se adresář `fastfoods`, který obsahuje souboru:

  `models.py` – zde vytvoříme modely pro db

  `views.py` – zde vytvoříme funkce

  `admin.py` – nastavení adminovské stránky

Django

- Vytvoříme modely – třídu v Pythonu, která bude reprezentovat tabulku v databázi
- Instance třídy bude reprezentovat záznam v databázi (řádek v tabulce)

```
class Restaurant(models.Model):
  name = models.CharField(max_length=30)
  opened = models.DateTimeField(default=datetime.now)
  capacity = models.IntegerField()
  company = models.ForeignKey('Company',
on_delete=models.CASCADE)
```

- Primární klíč je přidán automaticky

https://docs.djangoproject.com/en/3.1/ref/models/fields/#field-types

Django

- Vytvoříme databázové tabulky pomocí tzv. migrace

```
python3 manage.py makemigrations fastfoods
python3 manage.py migrate
```

- Názvy tabulek generovány automaticky na základě jména aplikace a modelu
- Primární klíč je vygenerován automaticky
- Pokud provedeme změny v tabulce, je třeba provést znova migraci

Django

- Interaktivní shell

  ```
  python3 manage.py shell
  ```

  - Můžeme přidat, mazat, editovat záznamy v db
- Adminovské webové rozhraní
  - Vytvořit tzv. `superuser` účet
  - Zaregistrovat modely v `admin.py`
  - přihlásit se přes `localhost:8000/admin/`

Django

- Interaktivní shell

```
python3 manage.py shell
```

`Restaurant.objects.get(id=1)` – vrací pouze jeden objekt

`Restaurant.objects.filter(id=1)` – vrací množinu objektů

- Filtrace (SELECT): `.filter(param__funkce=hodnota)`

```
Restaurant.objects.filter(name__contains='Po')
Restaurant.objects.filter(capacity__gt=15)
Restaurant.objects.filter(company__pk=1)
```

https://docs.djangoproject.com/en/3.1/ref/models/querysets/#field-lookups

Django

- Interaktivní shell

  ```
  python3 manage.py shell
  ```

- Mazání: `.delete`

  v modelu jsme u cizího klíče nastavovali parametr `on_delete`

- Možnosti:

  `CASCADE` Když smažeme záznam, na který se odkazuje cizím klíčem v jiné tabulce, jsou smazány i záznamy z druhé tabulky

  `SET_NULL` Atribut záznamu v jiné tabulce bude nastaven na NULL

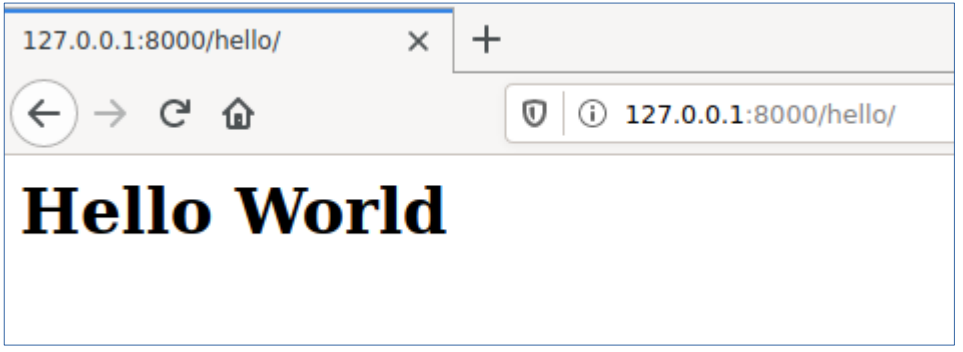
  `PROTECT (RESTRICT)` Nelze smazat záznam, pokud na něj odkazují záznamy v jiné tabulce

Django

- Views – funkce, pomocí kterých zobrazíme data na webu

```
from django.http import HttpResponse
    def hello_view(request):
        return HttpResponse('<h1>Hello, World</h1>')
```

- Poté určíme adresu, na které se tato funkce zobrazí
- `path('hello/', appname.views.hello_view)`



Django

- V URL je možné předávat v adrese hodnoty do view funkce

```
path('restaurant/<int:rest_id>/', appname.views.restaurant_info)
```

```
def restaurant_info(request, rest_id):
    r = Restaurant.objects.get(id = rest_id)
    return HttpResponse('<h1>Restaurant is {}</h1>'.format(r.name))
```

Vytvářet stránky takhle ve funkci samozřejmě není ideální
(datová a zobrazovací část měla být oddělena)

Proto existují templaty, kde nadefinujeme vzhled stránek
a z view tam pouze pošleme data, která chceme zobrazit

Django

View:

dict, obsahující proměnné,
které se předávají do templatu

```
def index(request):
  r = Restaurant.objects.all()
  return render(request, 'rest_index.html', {'rests': r})
```

Template (soubor `rest_index.html`):

```
<h1>List of restaurants</h1>
{% for rest in rests %} # konstrukce jazyka se uzavírají do {% %}
<li> {{ rest.name }} </li> # hodnoty proměnných se uzavírají do {{ }}
{% endfor %}
# v HTML nefunguje odsazení od okraje jako v Pythonu
proto je nutné cyklus ukončit (stejně tak podmínky)
```

Django

- Forms: zobrazíme HTML formuláře, které budou generovány automaticky pomocí Djanga
- Form je třeba vytvořit podobně jako model
- Např. formulář, který bude obsahovat dvě vstupní pole, kde budeme očekávat celočíselné hodnoty (validace formulářů probíhá v Djangu)

```
from django import forms
class CapacitySearchForm(forms.Form):
    from_capacity = forms.IntegerField(label='from')
    to_capacity = forms.IntegerField(label='to')
```

Django

- Co když chceme vytvořit formulář pro přidávání/editování záznamů v databázi – formulář tedy bude obsahovat prvky stejné, jako má model
- Nemusíme vytvářet nový formulář, ale použijeme `ModelForm` a řekneme, na základě kterého modelu se má formulář vygenerovat

```
class RestaurantForm(forms.ModelForm):
    class Meta:
        model = Restaurant
        # chceme zobrazit jen některé prvky?
        fields = ['capacity', 'name', 'company']
        # nebo chceme některé schovat? (buď jedno nebo druhé)
        exclude = [] # takhle zobrazíme všechny
```

Django

- Jak na obrázky?

```
settings.py:
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

models.py:
logo = models.ImageField(upload_to='images', blank=True)
```

poté je nutné provést migraci!

```
urls.py:
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [ …
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Django

- Jak na obrázky?

```
template:
<img src = "{{ model.logo.url }}" width=100 height=100>

<form action="{% url 'edit' form.instance.id %}" method="post"
enctype="multipart/form-data">

views.py:
form = MyForm(request.POST, request.FILES, instance = obj)
```