Python akademie Engeto

## Úvodní názvosloví

1. Datový typ

= kategorie dat, která určuje hodnoty a jejich typ

**Integer, int**

- celé číslo

- např. x = 5

**Float, float**

- desetinné číslo

- např. pi = 3.14

- odděluje se tečkou

**String, str**

- řetězec či sekvence znaků, psán v " "

- "Hello World"

**Boolean, bool**

- logické hodnoty

- True x False

2. Komplexní datový typ

**Seznam / list** **[ ]**

- kolekce hodnot, které mohou být různých typů a lze je měnit

- numbers [1, 2, 3, 4, 5]

- mixed [1, 2, 3, "Four", 5.8, False]

**N-tice / tuple ( )**

- podobné seznamům

- jsou neměnné (immutable)

- numbers (1, 2, 3, 4, 5)

**Slovník / dict { }**

- obsahují dvoje klíč : hodnota

- student = {"jmeno" : "Jan" , "vek" : "35"}

Set

Frozenset

Dictionary

3. Datový operátor

= vždy píšeme s mezerami

= priorita je dána matematicky, ale pro snadnou orientaci v kódu píšeme prioritní operace blíže k sobě

**+ Sčítání**

**- Odečítání**

**/ Dělení** *(u dělení je výsledek operace vždy float)*

**\* Násobení**

**// Celočíselné dělení**

- neboli kolikrát se druhé číslo vejde do prvního

- print(25 // 7) => 3

**% Modulo**

- neboli zbytek po dělení

- vezme hodnotu vpravo, tou vydělí hodnotu vlevo a to, co zbyde, vypíše jako modulo

- print(10 % 3) => 1

**== Porovnání hodnot**

- print(10 == 3) => False

- print(10 == 2\*5) => True

- pozor na porovnávání matematických operacích s floaty

**\*\* Square root**

- neboli kvadratický kořen / odmocnina

- print(16 \*\* 0.5) => 4.0 *(vrácená hodnota je vždy float)*

- můžeme zapsat i pomocí funkce **math.sqrt()**

1. import math

2. print(math.sqrt(16)) => 4.0*(vrácená hodnota je vždy float)*

Hierarchie datových operátorů

Pořadí Operátor Proces\_\_\_\_\_\_\_

1 ( ) závorky

2. \*\* umocňování

3. \* násobení

4. / dělení

5. + sčítání

6. - odčítání

**F-string**

= zkrácený zápis stringu s proměnnou

místo

print("Ahoj, hodnota a je", a, "a hodnota b je", b)

píšu

print(f"Ahoj, hodnota a je {a} a hodnota b je {b}")

1. LEKCE – datové typy a jejich metody

*Lektor: Valerie*

* Základem Pythonu jsou 3 pilíře:

1. **Datové typy** = int, float, str, bool,..
2. **Syntaxe** = funkce, podmínky, smyčky
3. **Knihovny** = decimal,…

# Datové typy

= každý objekt v Pythonu má ▪ datový typ

▪ hodnotu

▪ identitu *(umístění v paměti)*

## 1) Integer (int)

= celé číslo

* jak velký integer můžeme vytvořit? Omezení je pouze technické – dle velikosti paměti

## 2) Float (float)

= desetinné číslo

= odděluje se tečkou

- float se vytvoří, ať napíšu 3.14, nebo 3.0 nebo 3.

- má fixní místo v paměti = nemůže růst do nekonečna

**!** Operace s ním nejsou přesné kvůli desetinným místům bez zaokrouhlování

*print(0.1 + 0.3) => 0.400000008 a proto print(0.3 == 0.1 + 0.2) => False*

Porovnat float můžeme:

1. přes zabudovanou knihovu decimal
2. *import.decimal*
3. *print(decimal.Decimal(”0.1“) + decimal.Decimal(”0.2“) == decimal.Decimal(”0.3”) => True*
4. přes příkaz zaokrouhlení round (ve kterém navíc můžeme určit, na kolik míst chceme zaokrouhlit)

*print(round(0.1 + 0.2, 3) == 0.3) => True*

## 3) String (str)

= text a znaky (@, ., €, ?,…)

= vždy se píše do uvozovek, buď " " nebo ' '

**!** Problémové jsou kombinace citací, uvozovek a apostrofů v textu. Kombinujeme:

print("It’s his book") => It’s his book

print('Books name is "Hello World" ') => Books name is "Hello World"

Pro zápis It’s book is named "Hello World" postupujeme:

**\** escape character=> zapsání před speciální znak zajistí, že nebude brán jako

systémový

print("It’s book is named \"Hello World\" ") => It’s book is named "Hello World"

### **Speciální znaky stringů:**

**\‘** = apostrof

**\\** = zpětné lomítko

**\n** = zalomí text na nový řádek

**\r** = return carriage (vrácení na začátek řádku)

**\t** = tabulátor

**\b** = backspace

**\f** = form field (oddělovač k logickému oddělení textů)

**+** = operátor **+** není pro string sčítání, ale spojování znaků

**+** v integeru: print(2 + 2) => 4

**+** ve stringu: print("2" + "2") => 22

= nelze kombinovat str + int ani str + float

### **Další procesy stringů:**

Obsah obrázku text, Písmo, snímek obrazovky, číslo

Popis byl vytvořen automaticky

1. indexování stringu [ ]

= počítáme od 0, u stringu AUTOBUS je 0. písmeno A, 1. písmeno je U

= hodnotu indexu získáme zadáním čísla indexu do závorek []

print("Matouš"[1]) => a

print("Matouš"[0]) => M

**= pokud má string 6 hodnot, má index 0-5!**

print("Matouš"[5]) => š

print("Matouš"[6]) => ERROR

**= záporný index – poslední prvek, počítání od konce = zde končíme -1 (ne 0)**

print("Matouš"[-1]) => š

print("Matouš"[-6]) => M

1. Slicing stringu [ : ]

= vrátí nám několik hodnot z kolekce

= obsahuje dvě hodnoty, od : do = [1:10]

**!** hodnota do neznamená včetně, ale do hodnota -1

print("a.johanova@gmail.com"[2:7]) => johan

= pokud slicing začínáme od 0, můžeme provést zkrácený zápis:

v.1: print("a.johanova@gmail.com"[0:6]) => a.joha

v.2: print("a.johanova@gmail.com"[:6]) => a.joha

= záporný slicing zapíšeme následovně:

print ("a.johanova@gmail.com"[:-7]) => [a.johanova@gm](mailto:a.johanova@gm)

**!** chceme-li zjistit, zda je hodnota palindrom, použijeme zápis:

(slovo == slovo[::-1]) => True je palindrom, False není palindrom

1. Striding stringu [ : : ]

= krokování

= určuje, kterým krokem budeme vracet hodnoty z kolekce

kolekce=("1234567890")

print(kolekce[::1]) =>1234567890 (první a pak každý první prvek)

print(kolekce[::2]) => 13579 (první a pak každý druhý prvek)

print(kolekce[::4]) => 159 (první a pak každý čtvrtý prvek)

1. Kombinace slicingu a stridingu stringu [ : : ]

kolekce=("1234567890")

print(kolekce[1:8:2]) => 2468 (od indexu 1 do indexu 7 a každý druhý)

# 4) Změna datového typu

= print(type("2") => str

= print(type(2)) => int

**!** print(type(int("2"))) => int

* Text string nelze převést na int ani na float
* Float nelze prevést na string

# Proměnná v Pythonu

= něco, co nás odkáže na objekt

= pojmenování objektu

= je **měnitelná**

jmeno=("Matouš")

vek=(55)

print(jmeno,vek) => Matouš 55

Název proměnné nesmí:

* Začínat číslem
* Obsahovat speciální znaky
* Obsahovat mezery

Jak proměnou pojmenovat:

* MojeDatumNarozeni
* mojeDatumNarozeni
* moje\_datum\_narozeni *= nejpoužívanější, tuto používat*

# Range

= interval, rozsah

= umožňuje vytvořit **rozsah integeru od do**

= pracuje pouze s celými čísly

print(range(5))) => range(0,5) *(od, do, ale NE včetně)*

print(list(range(5))) => 0, 1, 2, 3, 4 *(#pro print seznamu je potřeba range převést na*

*seznam)*

= pokud range počítáme od 0, můžeme zapsat zkráceně jako range(5), jinak range (4,16)

= range můžeme krokovat stejně jako list a to rozšířením zápisu (4, 16, 2) = od 4, do 16 a pouze každý druhý prvek *(# start, stop, step)*

**Vytvoření range**

Počet argumentů v ( ) Význam

1 (5) Hodnota stop, rozsah 0 až stop-1

2 (1, 5) Hodnota start, stop

3 (1, 20, 2) Hodnota start, stop, step

= Range používáme uvnitř cyklu

For cislo in range(

2. LEKCE – sekvence a jejich metody

Lektor: Valerie

= sekvence jsou prvky, které obsahují dva a více datových prvků

## 1) List [ ]

= datový typ seznam, obsahuje **odkazy** na hodnoty

= kolekce hodnot je string, ale pro různé typy hodnot v kolekci je **list**

= má hranaté závorky (AltGr + F/G)

= oddělovačem hodnot je čárka

= obsah **lze po vytvoření měnit**

["Matouš", "Jan", "Marek"] ale i ["Matouš", 1900, 3.14]

= pro opakování lze použít operátor **\***

print["0"] \* 3 => 000

Prázdný seznam můžeme vytvořit následovně:

* prvni\_seznam = [ ] -> print(type) = 'list'
* druhy\_seznam = list() -> print(type) = 'list'

## 2) Tuple ( )

= neboli n-tice, obsahuje **odkazy** na hodnoty

= má kulaté závorky, ale může být i bez nich

= oddělovačem hodnot je čárka

= obsah **nelze po vytvoření měnit** (nelze změnit odkazy na hodnoty), ale pokud tuple odkazuje na seznam, tak odkaz změnit lze

Indexování v tuplu:

t = (2, 1.5, "Val")

t1 = (1, 2, 3)

print(t1[0]) => error

t = ([1, 2, 3], [1,2])

index 1 index 2

print(t[0][1]) => 2

index 0 hodnota 1

Změna hodnot v listu a tuplu:

l1 = [1, 2, 3]

l2 = l1

l2[0] = 6 -> print(l1,l2) => [6, 2, 3] [6, 2, 3]

Práce s listem a tuplem:

* můžeme s nimi pracovat stejně, jako se stringy:
  + **spojování**
  + **indexování**
  + **slicing**
  + **striding**
  + **opakování**
  + **rozšířené metody**

Převod listu na tuple

converted\_to\_tuple = tuple(l)

Indexování

**seznam = ["Matouš", "Lukáš", "Petr", "Jan", "Petr"]**

print(seznam[0]) => Matouš

print(seznam[0:2]) => Matouš, Lukáš

print(seznam[:3]) => Matouš, Lukáš, Petr

Slicing a striding listu

print(seznam[1:5:2]) => Lukáš, Jan

print(seznam[:2]) => Matouš, Lukáš

Slicing a striding a tuplu

**seznam = ("vývojáři", "admini", "office", "marketing", "helpdesk")**

print(t[1:4]) => admini, office, marketing

2. LEKCE – sekvence a jejich metody

Lektor: Valerie

Cyklus

= neboli smyčka

= anglicky 'loop'

= využití prakticky v každém skriptu

= cyklus je proces, který nám umožňuje vykonávat příkazy opakovaně, aniž bychom se museli ručně přepisovat

# 1) For cyklus

= také smyčka for, for loop

=> úkolem této smyčky je projít hodnotu (nebo hodnoty v proměnné) od první do poslední

* iterator = iterací se nám vrátí vždy jeden objekt
* iterace = smyčka (kolo) se pravidelně opakuje v jednotlivých iteracích (kolech)
* iterovatelný objekt = takový objekt, který lze pomocí smyčky procházet

**muj\_seznam = [1, 2, 3, 4, 5]**

* pokud chci vypsat každou hodnotu z listu, musela bych vyvolat print pro každou hodnotu zvlášť:

print(muj\_seznam[0]) => 1

print(muj\_seznam[1]) => 2

print(muj\_seznam[2]) => 3

…

* pro zjednodušení a efektivitu můžeme využít smyčku **for – in :**

**for nazev\_promenne in muj\_seznam:**

**print(nazev\_promenne)**

* + **for** = klíčový výraz pro předpis smyčky
  + **nazev\_promenne** = proměnná vytvořená pouze pro účely smyčky, její obsah se v každém kole smyčky vypíše
  + **in** = klíčový výraz, který říká, odkud mám brát hodnoty
  + **seznam** = iterovatelný zdroj hodnot
  + **print(nazev\_promenne)** = pokyn, co má smyčka vykonat

for cislo in muj\_seznam:

print(cislo) -> 1

2

3

4

5

**nebo:**

for cislo in muj\_seznam:

print(cislo)

print("Konec cyklu") -> 1

(= další kód) 2

3

4

5

Konec cyklu

**ale:**

for cislo in muj\_seznam:

print(cislo)

print("Další cyklus") 1

Další cyklus

2

Další cyklus

3

Další cyklus

4

Další cyklus

5

Další cyklus

**!** v podmínkách for-in ale i if-elif-else je velmi důležité dbát na správné odsazení

vnitřku cyklu/podmínky.

Iterovatelné datové typy

= datové typy, u kterých můžeme využívat smyčky

= jsou to ty, které se dají procházet

### string

for pismeno in "Matouš":

print(pismeno) => M

a

t

o

u

š

### list a tuple

for pismeno in ("a", "b", 18):

print(pismeno) => a

b

18

### dictionary

for klic in {"jmeno" : "Jan"}:

print(klic) => Jan

### set

for symbol in {"@", "?", "!"}:

print(symbol) => @

?

!

**Shrnutí**

**str ano int ne** (for i in 1, print i => error)

**list ano float ne**

**tuple ano**

**dict ano**

**set ano**

Metody datových typů v množině

= pro množiny můžeme využívat i jednotlivé metody datových typů

= např pro dictionary

**d = {"one" : 1, "two" : 2}**

* keys, values a items nám vrátí neseřazené hodnoty, ale vždy budou ve stejném pořadí
* o řazení rozhoduje Python náhodně

for k in d.keys():

print(k) => one

two

for v in d.values():

print(v) => 1

2

for (k,v) in d.items():

print(k,v) => one 1

two 2

Kombinace for cyklu, proměnné a podmínek

#### **Proměnná**

* příklad, součet sudých a lichých čísel ze seznamu:

seznam = [1, 2, 3, 4, 5, 6]

s1 = 0 #pro sudé

s2 = 0 #pro liché

for prvek in seznam:

if prvek % 2 == 0

s1 = s1 + prvek

else:

s2 = s2 + prvek

print(s1, s2)

=> 12 9

Rozšíření for, else (není if else!)

**for** = spouštěč iteratoru

**else** = po tom, co se spustí vše ve for, spustí se vše v else (třeba když neproběhne celý cyklus)

= není to buď anebo, ale nejdřív se pustí vše ve for a poté v else

Speciální ohlášení for cyklus

= smyčky můžeme doplnit ohlášeními, pokud potřebujeme přeskočit/pokračovat v průběhu cyklu

**a) BREAK**

= přeskočí zbytek smyčky a pokračuje kódem pod ní

for pismeno in "Matouš":

print(pismeno)

break *(# přeskočí další krok for cyklu)*

print("Ahoj")

= M

= pokud použiju break, tak mi proměnná (pismeno) zůstane s hodnotami, které získala do doby, než se zavolalo 'break' (pismeno "M")

for pismeno in "Matouš":

print(pismeno)

break

else:

print("konec smyčky")

print("Pokračujeme")

print(pismeno)

= M

Pokračujeme

M

**Kombinace:**

* for s ohlášením break
* for/else s ohlášením break
* for/else bez ohlášení brek se nevyužívá

=> break se většinou schová uprostřed if podmínky ve for cyklu

=> nejčastěji se používá for/else s ohlášením break a doplňujícím ohlášením pod cyklem

**pismena = ["a", "b", "c", "d", "e", "f", "g", "h"]**

for pismeno in písmena:

if pismeno == "e":

print("Mám hodnotu ->", pismeno)

break

else:

print("Nemám 'e' ")

print("Pokračuji v dalším kódu.")

= Mám hodnotu -> e

Pokračuji v dalším kódu

Ale pokud by "e" v seznamu nebylo, vyprintuje se:

= Nemám 'e'

Pokračuji v dalším kódu.

**b) CONTINUE**

= zastavuje iteraci, ale ne celý for cyklus, a přejde na další iteraci

= for/else v kombinaci s continue nefungují

**pismena = ["a", "b", "c", "d", "e", "f", "g", "h"]**

for pismeno in písmena:

if pismeno in {"a", "b", "c", "d"}: *(# v setu pismen)*

continue

print("Hodnota je:", pismeno)

= Hodnota je e

Hodnota je f

Hodnota je g

Hodnota je h

= pokud je v proměnné písmeno a, b, c nebo d, Nepokračuj k printu, ale vrať se zpět k if

= pokud je v proměnné jiné písmeno, než a, b, c nebo d, for cyklus se vyprintuje

**Kombinace:**

* lze kombinovat continue, break a else
* pokud je ale v zápisu break, else se nevyhodnotí

continue, break, else:

for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

if i < 5:

continue

print(i)

if i == 9:

break *(#hodnota je True)*

else:

print("else se spustil")

= 5

6

7

8

9

continue, else:

for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

if i < 5:

continue

print(i)

if i == 59:

break *(#hodnota je False)*

else:

print("else se spustil")

= 5

6

7

8

9

else se spustil

další příklad, počet sudých čísel:

s = 0

for i in range(1000):

if i % 2 != 0:

continue

s += s

if i == 10:

break

print(s)

= 6

další příklad, součet lichých čísel:

s = 0

for i in range(1000):

if i % 2 != 0:

continue

s += i

if i == 10:

break

print(s)

= 30

**c) PASS**

**= nic nedělá**

**= dočasný placeholder, díky kterému Python nevyvolá syntaktickou chybu**

**for pismeno in "Matouš":**

**pass**

**print("Ahoj")**

**= Ahoj**

**d) NESTOVANÝ FOR LOOP**

= for cyklus vložen do jiného for cyklu

= teoreticky může být nekonečno úrovní for cyklů v sobě, ale nedoporučuje se to, protože zpomaluje kód

= ideální jsou 2 vrsty, maximálně 3

= pozor, proměnná v iteraci 2/3 musí mít jiný název

= iterace v iteraci = iterace2

= iterace v iteraci v iteraci = iterace3

l = [1, 2, 3]

for i in l:

for j in l:

print(i,j)

=

1 1

1. 2
2. 3
3. 1

2 2

2 3

3 1

3 2

3 3

obsah\_csv = [

["jmeno; prijmeni; email; projekt"], *#index 0*

["Matouš; Holinka; [m.h.@gmail.com](mailto:m.h.@gmail.com); HR"], *#index 1*

["Petr; Svetr; [p.s.@seznam.cz](mailto:p.s.@seznam.cz); Fullstack"] *#index 2*

]

for radek in obsah\_csv:

print("===")

print(radek[0].split(";")) *(#pro každý index rozděl string podle středníku)*

=>

===

jmeno, prijmeni, email, projekt

===

Matouš, Holinka, [m.h@gmail.com](mailto:m.h@gmail.com), HR

===

Petr, Svetr, [p.s.@seznam.cz](mailto:p.s.@seznam.cz), Fullstack

+ for bunka in (radek[0].split(";")):

print(bunka)

=>

jmeno, prijmeni, email, projekt *(# procházení řádku)*

jmeno *(#procházení sloupce)*

prijmeni

email

projekt