



OSNOVA

< PŘEHLED KURU

54% hotovo z Lekce 4

Vyhledávat



Obsah a prerekvizity

ONLINE PYTHON AKADEMIE / FOR CYKLUS / OBSAH A PREREKVIZITY

Po prvních třech lekcích bys měl mít základní znalosti ohledně:

- některých datových typů: `int`, `float`, `str`, `list`, `tuple`, `set`, `dict` jejich metody a operace s nimi,
- rozhodování pomocí `if`, `elif` a `else`,
- práce v tebou vybraném IDE (příp. editoru)

Jak psát vlastní kód

ONLINE PYTHON AKADEMIE / FOR CYKLUS / JAK PSÁT VLASTNÍ KÓD

Na úvod bychom zmínili několik věcí, které pomůžou jak tobě, tak ostatním, kteří budou tvůj zápis číst.

Čitelný kód

Ať už jsi přišel z našeho začátečnického kurzu, nebo ne, měl by jsi vědět následující. V přechozím kurzu jsme se hodně soustředili na základní syntax a koncepty. Nyní je tedy ten pravý čas se dozvědět něco o **PEP-8**. O co jde? PEP-8 jsou doporučení standartů, jak formátovat kód tak, aby byl **čitelný, konzistentní a snadno se do budoucna udržoval**. PEP-8 sice nemusíš dodržovat, aby tvůj kód fungoval, ale programátoři tyto doporučení následují. Ulehčí si tím totiž vzájemně práci.

Vše co potřebuješ vědět o PEP-8 můžeš najít na pep.org nebo python.org.

Vzájemná solidarita

Tvým cílem by tedy mělo být psát **kód, který není srozumitelný pouze tobě, ale i ostatním**. Ušetříš práci i nám. Pokud bude tvůj kód přehledný, snadno čitelný, správně formátovaný a strukturovaný, ušetříš nám práci při jeho kontrole. Na základě toho ti potom můžeme i poskytnout **lepší feedback** :)

A to platí i do budoucna, kdy už budeš ostřílený programátor a budeš třeba pracovat v týmu vývojářů. Tvý kolegové určitě ocení snadnou práci s tvým kódem a ty s jejich.

Komentáře

Na závěr bychom rádi zmínili, že v našem kód častu vidíš vcelku podrobné komentování kódu. Ve výuce programování je to opodstatněné. Chceme, aby bylo vše naprostě jasné. Ve svém kódu určitě **komentáře dále použivej, ale jen pokud jsou opodstatněné**.

Kód by se obecně měl psát tak, aby byl pochopitelný už z jeho zápisu. Například ze jména funkce by jsi měl na první dobrou pochopit, co funkce dělá. A to platí i pro člověka, který tvůj kód ještě neviděl. Může ale samozřejmě nastat situace, že i přes přehledný, čitelný a čistý zápis je komentář potřeba - v tom případě do toho :)

Kód bloky

ONLINE PYTHON AKADEMIE / FOR CYKLUS / KÓD BLOKY

Aby jsi se lépe orientoval na našem kurzu, měl by jsi vědět, že můžeš narazit na 2 druhy bloků kódu.

1. Python Interactive Session

Jedná se vlastně o práci s Python kódem příkazovém řádku. Blok bude černý a každý řádek zde bude začít s `>>>`.

```
>>> print('Hello')
Hello
```

Python by jsi měl už nastavený a být schopen s ním tedy pracovat v příkazovém řádku. Pro připomenutí, stačí napsat 'Python' a

zmacknout enter.

Bude dobré, když budeš mít vždy otevřený jak příkazový řádek, tak editor kódu. Když budeme pracovat s příkazovým řádkem, pracuj s příkazovým řádkem. Když budeme pracovat s editorem, pracuj také s editorem :)

Při psaní v příkazovém řádku stačí vždy napsat kód a spustit enterem. Python ti automaticky vrátí výstup kódu.

```
>>> 5 + 5  
10
```

2. Python skript

Python skripty nebudou obsahovat `>>>` na začátku řádku. Budeme je psát v editoru kódu, do Python souboru, který uložíme s koncovkou `.py`. Tyto bloky budou bílé:

```
1 print('Hello Python')
```

Psání kódu v editoru znamená, že můžeme kód také spustit **přímo v editoru**, jehož výsledek bude v černém bloku:

```
'Hello Python'
```

Někdy možná také budeme chtít spustit náš skript v příkazovém řádku. Poznáš to tak, že kód blok bude černý a bude začínat buď znakem dolar `$`, nebo porovnávacím operátorem `>`:

```
$ python hello_python.py  
'Hello Python'
```

Tento kód znamená, že chceme, aby jsme spustili skript, který je uložen v souboru `hello_python.py`.

Objekty

ONLINE PYTHON AKADEMIE / FOR CYKLUS / OBJEKTY

Na úvod tohoto kurzu si povíme, co přesně v Pythonu znamená slovo **objekt**. Python je totiž tzv. **objektově orientovaný jazyk**. I když se v tomto kurzu nebudeme věnovat objektově orientovanému programování (OOP), vyplatí se nám pochopit koncept **objektů**. Už jen proto, že toto slovo budeme občas v kurzu používat.

Co je to objekt?

V Pythonu je objekt vlastně skoro všechno. Jediné, co mezi objekty neřadíme, jsou konstrukce, které říkají, jak mají konkrétní objekty vypadat. Takovou konstrukci nazýváme třída - **class**. Mezi konstrukci můžeme zařadit například datové typy. Ty nám určují, jak konkrétní objekty vypadají. Proto výstup funkce `type` vždy říká, že něco je class. Vysvětlíme si to. Například class datového typu integer (celé číslo) je int:

```
1 # Vytvoreni promenne a prirazeni celeho cisla '1'  
2 cislo = 1  
3  
4 # Zjistovani dane 'class'  
5 print( type(cislo) )
```



```
<class 'int'>
```

Pro zajímavost si můžeš vyzkoušet, jaký má výstup funkce `type`.

Objektům se věnujeme v samostatném kurzu, který se zaměřuje pouze na OOP. Paradigma objektově orientovaného programování je pro mnohé těžko uchopitelné, pokud nemají dobré základy. Nám zatím postačí, když budeme vědět, že:

- v Pythonu je objekt skoro vše, až na třídy/konstrukce/class,
- **třída** je proto **konstrukce** daného objektu,
- **objekt** obsahuje **konkrétní data** a je uložený na konkrétním místě v paměti.

Tvoje úloha

Než spustíš kód niže, zkus si tipnout, zda jednotlivé funkce `print()` vrátí hodnotu **True**, nebo **False**

```
1 # Zadani promennych  
2 a = [1, 2, 3]  
3 b = a  
4 c = list(a)  
5
```

```
6 # Tisk promennych
7 print(a == b)
8 print(a is b)
9 print(a == c)
10 print(a is c)
```

SPUSTIT ZNOVU

```
True
True
True
False
```

Rozbor

- Funkce `id()` vrací **identifikační číslo** ukazující na místo v paměti, kde jsou uloženy údaje o objektu.
- Znaménkem "==" se v podstatě ptáš, jestli mají dva objekty stejnou hodnotu (vlastnosti).
- Klíčové slovo "is" porovnává, jestli se jedná o jeden a ten samý objekt.
- `a == b` vrací True, protože obě proměnné mají stejnou hodnotu.
- `a is b` vrací True, protože obě proměnné **odkazují** na stejný objekt v paměti.
- `a == c` vrací True, protože (opět) obě proměnné mají stejnou hodnotu.
- `a is c` vrací False, protože proměnná 'c' **odkazuje** na nově vzniklý objekt, který byl vytvořen funkcí `list()`.

Co tě čeká v této lekci

ONLINE PYTHON AKADEMIE / FOR CYKLUS / CO TĚ ČEKÁ V TÉTO LEKCI

Tento kurz začneme lekcí o datovém typu `range` a smyčkou `for`.

U `range` se dozvímě:

- jeho princip,
- jak s ním pracovat pomocí operací různých operací (indexing, membership testing, apod.)
- jeho užití

U smyčky `for` se dozvímě:

- její princip,
- její užití v kontextu různých datových typů,
- práce s `continue` a `break`,
- a další..

DATOVÝ TYP RANGE

Range - vytváření a princip

ONLINE PYTHON AKADEMIE / FOR CYKLUS / DATOVÝ TYP RANGE / RANGE - VYTVAŘENÍ A PRINCIP

Range je **neměnný/immutable** datový typ, stejně jako **string nebo tuple**. Jediný způsob, jakým je možné vytvořit objekt range je použít funkci `range()`.

Poznámka: Range patří mezi sekvenční datové typy (str, tuple, list).

Range argumenty

Existují 3 způsoby, jak můžeme vytvořit range:

1. `range(stop)`
2. `range(start,stop)`
3. `range(start,stop,krok)`

Argumenty funkce `range(start,stop,krok)` musí být celá čísla - **integer**.

- **stop** - Číslo, u kterého by měla sekvence skončit. Konečné číslo nebude vygenerováno. Pokud není specifikován argument **start**, range vygeneruje sekvenci od integer 0. Pokud napišu `range(10)`, sekvence čísel je od 0 do 9 včetně (bez 10).
- **start** - Pokud je argument **start** specifikován, musí být definován i argument **stop**. Sekvence range nemusí začínat na 0, ale může začít na jakémkoli celém čísle, které je specifikováno argumentem **start**.
- **krok** - Umožní vygenerovat každý n-tý integer mezi **start** a **stop**. Když napišeme `range(0,10,2)`, můžeme vygenerovat když druhý prvek sekvence od 0 do 10. Pokud je **krok** záporné číslo, sekvence je vygenerována pozpátku - `range(10,0,-1)` vygeneruje čísla od 10 do 1.

Vygenerování čísel do listu

Všimni si, že když používáme záporný krok, první číslo musí být větší. Co by se stalo, kdybychom zkusili vytisknout `**range(0,10,-1)` ???

```
1 # Tisk klesajici sekvence pomocí range
2 print(list(range(1,10,-1)))
```

```
[]
```

Dostali bychom prázdny list.

Vygenerování čísel bez listu?

Pokud chceme čísla vygenerovaná pomocí `range()` vytisknout, musíme tuto funkci vložit do listu - `list()`. Následně použijeme funkci `print()`, to už znáš. Co by se stalo, kdybychom tak neudělali?

```
1 # Tisk klesajici sekvence pomocí range
2 print(range(1,10,-1))
```

```
range(1, 10, -1)
```

Dostali bychom zpět objekt range, nikoliv vygenerovaná jednotlivá čísla.

Vygenerování čísel do tuplu

Objekt range můžeme kromě listu vložit také do tuplu - `tuple()`.

```
1 # Tisk klesajici sekvence pomocí range
2 print(tuple(range(10)))
```

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Range tedy generuje čísla tzv. "na požádání", a ne rovnou. Když tedy vložíme objekt range do funkce print, nevidíme celou sekvenci generovaných čísel. Pokud chceme číslo v jednom kroku vygenerovat, můžeme použít funkce `list()` nebo `tuple()`. Důvod, proč tomu tak je, se dozvídá v našich pokročilejších kurzech, když probíráme například koncept *generátorů*. Prozatím bychom měli vědět, že tisk pouhého `range()` objektu nevytiskne sekvence čísel, kterou generuje.

Experimentování s range() argumenty

ONLINE PYTHON AKADEMIE / FOR CYKLUS / DATOVÝ TYP RANGE / EXPERIMENTOVÁNÍ S RANGE() ARGUMENTY

Nyní si spust' kód, který obsahuje všechny možné variace range. Tohle ti pomůže pochopit, jak se range chová při různých vstupech.

```
1 # 1. od > do
2 print("1.", list(range(5,1)))
3
4 # 2. do == 0
5 print("2.", list(range(0)))
6
7 # 3. od < do a krok < 0
8 print("3.", list(range(3,9,-1)))
9
10 # 4. od > do a krok < 0
11 print("4.", list(range(10,0,-1)))
12
13 # 5. krok > 2
14 print("5.", list(range(0,10,3)))
15
16 # 6. krok < -1
17 print("6.", list(range(10,0,-2)))
18
19 # 7. krok == 0
20 print("7.", list(range(1,10,0)))
```

SPUSTIT ZNOVU

```
1. []
2. []
3. []
4. [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
5. [0, 3, 6, 9]
6. [10, 8, 6, 4, 2]
```

Výsledek

```
1. []
2. []
3. []
4. [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
5. [0, 3, 6, 9]
6. [10, 8, 6, 4, 2]
Traceback (most recent call last):
  File <stdin>, line 20, in <module>
    print("7.", list(range(1,10,0)))
ValueError: range() arg 3 must not be zero
```

Na počtu argumentů range() záleží

ONLINE PYTHON AKADEMIE / FOR CYKLUS / DATOVÝ TYP RANGE / NA POČTU ARGUMENTŮ RANGE() ZÁLEŽÍ

Jak range vyhodnotí obdržený argument? Na základě počtu specifikovaných argumentů:

Případ	Příklad	Význam argumentu	Výsledek
1 argument	<code>range(5)</code>	vždy DO (stop)	0,1,2,3,4
2 argumenty	<code>range(3,5)</code>	vždy OD a DO (start, stop)	3,4
3 argumenty	<code>range(3,12,2)</code>	vždy OD, DO, a KROK (start, stop, krok)	3,5,7,9,11

Tvoje úloha

- Na řádku 2 vytiskni range, který ti vygeneruje každý prvek v rozmezí od 0-19.
- Na řádku 5 vytiskni range, který ti vygeneruje každý prvek v rozmezí od 15-19.

- Na řádku 8 vytiskni range, který ti vygeneruje *každý třetí prvek v rozmezí od 10-19*.
- Do proměnné *nums* na řádku 11 vytvoř list pomocí range, který bude obsahovat každé číslo, které je dělitelné 5 v rozmezí od 1-100.
- Na řádku 14 vytiskni proměnnou *nums*.

```

1 # Kazdy prvek rozmezi 0-19
2 print(list(range(20)))
3
4 # Kazdy prvek rozmezi 15-19
5 print(list(range(15,20)))
6
7
8 # Kazdy treti prvek rozmezi 10-19
9 print(list(range(10,20,3)))
10
11
12 # Promenna nums
13 nums = list(range(5,101,5))
14
15 # Tisk promenne nums
16 print(nums)

```

SPUSTIT ZNOVU

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[15, 16, 17, 18, 19]
[10, 13, 16, 19]
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100]
```

Řešení

V rozbalovacím boxu najdeš řešení úlohy s vysvětlením.

Zobrazit řešení

```

1 # Kazdy prvek rozmezi 0-19
2 print(list(range(20)))
3
4 # Kazdy prvek rozmezi 15-19
5 print(list(range(15,20)))
6
7 # Kazdy treti prvek rozmezi 10-19
8 print(list(range(10,20,3)))
9
10 # Promenna nums
11 nums = list(range(5,101,5))
12
13 # Tisk promenne nums
14 print(nums)

```

Indexing

ONLINE PYTHON AKADEMIE / FOR CYKLUS / DATOVÝ TYP RANGE / INDEXING

Stejně jako u stringu, listu, či tuplu, i u range existuje určité pořadí. Rychlé opakování - každý prvek má určitou pozici, která se nazývá **index**. Pomocí indexu můžeme získat prvek na dané pozici, čemuž říkáme **indexování - indexing**. Důležité je, že **první pozici má index 0**.

```

1 # Ziskani cisla na indexu 0
2 print(range(10)[0])
3
4 # Vysledek
5 0

```

```
6  
7 # Ziskani cisel na indexu 5  
8 print(range(10)[5])
```

```
5
```

Slicing a Striding

ONLINE PYTHON AKADEMIE / FOR CYKLUS / DATOVÝ TYP RANGE / SLICING A STRIDING

Pomocí **slicingu** dostáváme řez, **slice**, tedy několik prvků v intervalu `[start:stop]`.

Měli bychom znát jsme 4 způsoby, jak provést **slicing**:

- `sekvice[start:stop]`
- `sekvice[:stop]`
- `sekvice[start:]`
- `sekvice[:]`

Co ale vlastně funkce `range()` vraci?

```
1 # Ziskani slicu z range  
2 print(range(10)[2:6])
```

```
range(2, 6)
```

Funkce `range()` vrací tzv. range objekt. Kdybychom chtěli s hodnotami dále operovat, mohli bychom převést range třeba na list, a to pomocí funkce `list()`.

```
1 # Ziskani slicu z range  
2 print(list(range(10)[2:6]))
```

```
[2, 3, 4, 5]
```

Striding odkazuje na rozšíření operace slicing použitím **třetí hodnoty** v hranačních závorkách - **krok**. Ten říká Pythonu, kolik prvků by mělo být přeskoučeno při výběru. Tato hodnota je **defaultně** nastavena na **1**. Striding se zapisuje:

- `sekvice[start:stop:krok]`

```
1 # Ziskani kazdeho 3 prvku  
2 print(list(range(10)[::3]))
```

```
[0, 3, 6, 9]
```

Membership testing

ONLINE PYTHON AKADEMIE / FOR CYKLUS / DATOVÝ TYP RANGE / MEMBERSHIP TESTING

V kurzu pro začátečníky jsme zmíňovali tzv. **membership test**. Řekli jsme si, že se používá pro zjištění, zda se určitý prvek nachází v dané sekvenci (string, list, tuple). Zjišťovali jsme to pomocí operátora `in`.

Stejně jako třeba u stringu, membership test lze použít i u datového typu range pro zjištění, zda se v něm nachází určité **celé číslo (integer)**.

```
1 # Membership test  
2 print(6 in range(10))
```

```
True
```

Poznámka: Pozor, přitomnost lze testovat pouze u celých čísel.

```
1 # Membership test  
2 print(6.2 in range(10))
```

```
False
```

Konverze

ONLINE PYTHON AKADEMIE / FOR CYKLUS / DATOVÝ TYP RANGE / KONVERZE

Range je specifický v tom, že přijímá pouze **celá čísla (integers)**. Neumožňuje tedy konverzi z jiných datových typů na range:

```
>>> range([1,2,3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'list' object cannot be interpreted as an integer
```

Nepodporované Operace

ONLINE PYTHON AKADEMIE / FOR CYKLUS / DATOVÝ TYP RANGE / NEPODPOROVANÉ OPERACE

Range **nepodporuje spojování** (concatenation) nebo **opakování** (repetition).

Concatenation:

```
>>> r = range(10)
>>> list(r+r)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'range' and 'range'
```

Repetition:

```
>>> r * 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for *: 'range' and 'int'
```

Kde se range používá

ONLINE PYTHON AKADEMIE / FOR CYKLUS / DATOVÝ TYP RANGE / KDE SE RANGE POUŽÍVÁ

Objekty range nejčastěji používáme ke **generování sekvencí celých čísel**. Tyto sekvence poté můžeme použít ve smyčkách **for**.

O smyčkách **for** si povíme v následující sekci.

SMYČKA FOR

Syntaxe

ONLINE PYTHON AKADEMIE / FOR CYKLUS / SMYČKA FOR / SYNTAXE

Smyčka for (angl. for loop) umožňuje projít sekvenci objektů. For loop funguje jen na tzv. **iterovatelné Python objekty** - například sítíky/sekvence, jako jsou tuple, listy, slovníky, stringy, range, apod.

Zápis for loop

```
1 for prvek in iterable:  
2     prikaz
```

Složení for loop smyčky

- Klíčové slovo `for`,
- jméno proměnné (v našem příkladu `prvek`),
- klíčové slovo `in`,
- iterovatelný objekt (v našem příkladu `iterable`, což může být např. `range(10)`),
- dvojtečka `::`.

Tvoje úloha

Než spustíš skript níže, zkus uhodnout, co bude výsledkem.

```
1 # For loop  
2 for num in range(10):  
3     print(num)
```

SPUSTIT ZNOVU

```
0  
1  
2  
3  
4  
5
```

Princip smyčky

ONLINE PYTHON AKADEMIE / FOR CYKLUS / SMYČKA FOR / PRINCIP SMYČKY

- Smyčka For prochází jednotlivé prvky podle jejich pořadí (v případě sekvencí), nebo nahodile (v setech a slovnících).
- **Prochází** znamená, že odkaz daného prvku, který je na řadě, je uložen v proměnné, která je v hlavičce (v příkladu níže jsme ji pojmenovali `znak`). Proměnou v hlavičce můžeme nazvat libovolně.
- Prvek, který je uložený v hlavičce proměnné (`znak`), může být poté použit uvnitř těla smyčky. V našem příkladu převádíme každé písmeno - buď na velké, nebo malé.
- Smyčka se opakuje jen tolikrát, kolik máme prvků uvnitř procházeného objektu. Zde budeme procházet každé písmeno stringu '`Python`' a po projití každého písmene se smyčka sama ukončí.

```
1 # Vytvoreni prazdneho stringu pro ukladani  
2 konvertovane_slovo = ''  
3  
4 # Hlavicka smycky for  
5 for znak in 'Python':  
6     # Podminky  
7     if znak.isupper():  
8         konvertovane_slovo = konvertovane_slovo + znak.lower()  
9     else:  
10        konvertovane_slovo = konvertovane_slovo + znak.upper()  
11
```

```
12 # Tisk vysledku  
13 print(konvertovane_slovo)
```

pyTHON



Pro vizualizaci příkladu výše můžeme použít [Python Tutor](#). Stačí zkopiřovat a vložit kód výše a potom kliknout na "Visualize Execution".

Příklad výše by se dal verbalizovat takto: Pro (`for`) každý charakter (dočasně uložen v proměnné 'znak') uvnitř stringu 'Python' převed' znak na malé, nebo velké písmeno.

Iterace range()

ONLINE PYTHON AKADEMIE / FOR CYKLUS / SMYČKA FOR / ITERACE RANGE()

S `range()` - Range objekty (datové typy) reprezentují sekvence celých čísel. To nám umožňuje je procházet a použít dané číslo k získání indexu prvků jiné sekvence.

```
>>> muj_str = 'Hello World'  
>>> for i in range(len(muj_str)):  
...     print(muj_str[i])  
H  
e  
l  
l  
o  
W  
o  
r  
l  
d
```

Bez `range()` - V kódu výše generujeme range od 0 do posledního indexu stringu `muj_str`. Funkci `len()` používáme ke zjištění délky stringu. Tento zápis `range(len(objekt))` ale není zrovna srozumitelný. Proto bylo lepší použít `for loop` bez range.

```
>>> for i in my_str:  
...     print(i)  
H  
e  
l  
l  
o  
W  
o  
r  
l  
d
```

Python ví, jak procházet sbírkou prvků v `muj_str`. Nemusíme mu tedy poskytnout čísla indexů.

Iterace sekvencí a setů

ONLINE PYTHON AKADEMIE / FOR CYKLUS / SMYČKA FOR / ITERACE SEKVENCÍ A SETŮ

Ať procházíme listy, tuply, sety nebo range, hlavička smyčky for by měla obsahovat **jednu proměnnou**, ve které uchováváme dané prvky.

```
>>> for cislo in range(10):  
...     print(cislo, cislo**2)  
0 0  
1 1  
2 4  
3 9  
...  
9 81
```

Dokud nezklikneme více proměnných (cislo i), než je třeba, ohueví se **chubové klášení**.

```
>>> for cislo,i in range(10):
...     print(cislo, cislo**2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

Trik

Nicméně tuple, který obsahuje jiné tuply o 2 prvcích, můžeme výše uvedeným způsobem projít.

```
>>> data = (('Věk',43),('Jméno','John'),('Příjmení','Smith'))
>>> for kategorie, hodnota in data:
...     print(kategorie, ':', hodnota)
Věk : 43
Jméno : John
Příjmení : Smith
```

Opět můžeme použít [Python Tutor](#) k vizualizaci výše uvedeného kódu.

Obdobným způsobem se potom v praxi dají najednou projít x, y souřadnice pixelu určitého obrázku. Když takhle získáme každý pixel obrázku, můžeme jej třeba obrátit.

Tvoje úloha

Byla by možné projít tuple, který se skládá z 3členných tuplů? Jak bychom to udělali?

```
1 # Zadání tuplu
2 data = (('Věk',43,True),('Jméno','John',True),('Příjmení','Smith',False))
3
4 # For loop
5 for kategorie, hodnota, boolean in data:
6     print(f'{kategorie} : {hodnota} --> {boolean}')
```

SPUSTIT ZNOVU

```
Věk : 43 --> True
Jméno : John --> True
Příjmení : Smith --> False
```

Řešení

V rozbalovacím boxu najdeš řešení úlohy s vysvětlením.

Zobrazit řešení

```
1 # Zadání tuplu
2 data = (('Věk',43,True),('Jméno','John',True),('Příjmení','Smith',False))
3
4 # For Loop
5 for a,b,c in data:
6     print(a,b,c)
```

```
Věk 43 True
Jméno John True
Příjmení Smith False
```

Iterace slovníků

ONLINE PYTHON AKADEMIE / FOR CYKLUS / SMYČKA FOR / ITERACE SLOVNÍKŮ

U slovníku

```
1 zamestnanci = {'Jméno': 'John', 'Příjmení': 'Smith', 'Věk': 43}
```

... využijeme našich znalostí **metod slovníku**, abychom prošli klíče, hodnoty nebo obojí.

Pouze klíče

Abychom prošli jednotlivé klíče slovníku, stačí napsat název slovníku na místo sbírky, kterou procházíme.

```
1 # For loop
2 for klic in zamestnanci:
3     print(klic)
4
5 # Výsledek
6 Věk
7 Jméno
8 Příjmení
```

To znamená, že procházení klíčů je nastavené u slovníku jako defaultní možnost.

Pouze hodnoty

Když už víme, jak procházet klíče, můžeme se naučit procházet jeho **hodnoty**. K tomuto účelu použijeme metodu `.values()`.

```
1 # For Loop
2 for hodnota in zamestnanci.values():
3     print(hodnota)
4
5 # Výsledek
6 43
7 John
8 Smith
```

Klíče i hodnoty

Nakonec chceme získat klíč i hodnotu. V tomto případě musíme:

- poskytnout **2 proměnné v hlavičce smyčky loop**, kde uchováme klíč a hodnotu,
- použít metodu `.item()`.

```
1 # For Loop
2 for klic, hodnota in zamestnanci.items():
3     print(klic, hodnota)
4
5 # Výsledek
6 Věk 43
7 Jméno John
8 Příjmení Smith
```

Iterace pozpátku

ONLINE PYTHON AKADEMIE / FOR CYKLUS / SMYČKA FOR / ITERACE POZPÁTKU

Je nám asi už jasné, jako projít objekt. Nyní si ukážeme, jako ho projít pozpátku.

Mohli bychom to udělat například takto:

```
1 # Promenna
2 muj_str = 'Python'
3
4 ## End loop
```

```
4 # For Loop
5 for index in range(len(muj_str)-1,-1,-1):
6     znak = muj_str[index]
7     print(znak, end='')
```

nohtyP

Ale opět, kód výše není zrovna jako učebnice Pythonu. Aby vypadal lépe, mohli bychom použít tzv. **built-in funkci** (funkce, která je součástí základní knihovny Pythonu) **reversed()**.

```
1 # Promenna
2 muj_str = 'Python'
3
4 # For Loop
5 for znak in reversed(muj_str):
6     print(znak, end='')
```

nohtyP

Druhý zápis je více čitelný, šetří místo a také jeho provedení je rychlejší, protože funkce **reversed()** je optimalizována pro tyto účely. Funkce **reversed()** uspořádá původní objekt pozpátku.

Je také důležité říct, že funkce **print()** nyní obsahuje nový input - **end=''**. **Tohle způsobí, že každé další písmeno bude vytisknuto na stejný řádek.**

Tvoje úloha

Napiš loop (smyčku), která vytiskne string **'New York'** obráceně na jeden řádek.

```
1 # String
2 muj_str = 'New York'
3
4 # For loop
5 for pismeno in reversed(muj_str):
6     print(pismeno,end='')
```

SPUSTIT ZNOVU

kroY weN

Řešení

V rozbalovacím boxu najdeš řešení úlohy s vysvětlením.

Zobrazit řešení

```
1 # String
2 muj_str = 'New York'
3
4 # For Loop
5 for znak in reversed(muj_str):
6     print(znak, end='')
```

Vnořené smyčky

ONLINE PYTHON AKADEMIE / FOR CYKLUS / SMYČKA FOR / VNOŘENÉ SMYČKY

Nyní si představme, že máme tabulku v Excelu. Každá tabulka má řádky a sloupce. Abychom odkázali na libovolnou buňku, potřebujeme použít číslo řádku a sloupce, kde se buňka nachází. Těmto číslům se říká souřadnice.

Kdybychom chtěli (a tak to i v praxi je), mohli bychom reprezentovat každou tabulku jako list listů. Vezměme následující tabulku a převeďme ji do listu listů.

	0	1	2	3
0	ID	Name	Price	Amount
1	X131	Pipe	2.05	1000
2	XT12	Screw	0.35	1000
3	Z43	Nail	0.95	1000
4	P843	Tape	1.39	1000

Tabulku výše můžeme reprezentovat jako list listů následovně (každý řádek reprezentuje jeden list):

```
1 tabulka = [['ID', 'NAME', 'PRICE', 'AMOUNT'],
2             ['X131', 'Pipe', 2.05, 1000],
3             ['XT12', 'Screw', 0.35, 1000],
4             ['Z43', 'Nail', 0.95, 1000],
5             ['P843', 'Tape', 1.39, 1000]
6 ]
```

Abychom prošli jednotlivé prvky v každém z listů, budeme potřebovat použít smyčku for, a to dvakrát (jedna je vnořena do druhé). Spusť si následující kód a podívej se na výsledek.

```
1 # Tabulka
2 tabulka = [['ID', 'NAME', 'PRICE', 'AMOUNT'],
3             ['X131', 'Pipe', 2.05, 1000],
4             ['XT12', 'Screw', 0.35, 1000],
5             ['Z43', 'Nail', 0.95, 1000],
6             ['P843', 'Tape', 1.39, 1000]
7 ]
8
9 # For loop
10 for radek in tabulka:
11     for prvek in radek:
12         print(prvek)
```

SPUSTIT ZNOVU

```
ID
NAME
PRICE
AMOUNT
X131
Pipe
```

Poznámka: Vnořené smyčky jsou často používány pro práci s vnořenými sbírkami dat - list, slovník, atp.

Smyčka for nemusí procházet kód nebo objekt až do konce. Konec iterace můžeme kontrolovat pomocí klíčových slov **continue** a **break**. Obě slova se obvykle používají ve spojení podmínkovým výrazem:

- Můžeme přeskočit k hlavičce smyčky for a pokračovat s dalším prvkem (**continue**),
- můžeme přeskočit zbytek prvků a pokračovat v kódu, který pokračuje pod smyčkou for (**break**).

Continue a break mohou být použity jak ve smyčce while, tak ve smyčce for. Pojdme si je rozebrat trochu více.

Continue podrobně

Když narazíme ve smyčce loop na *continue*, kód skočí zpět na začátek smyčky a pokračuje dál:

- ve smyčce **for** jde Python na další prvek v procházené sbírce,
- ve smyčce **while** se Python vrátí do hlavičky a vyhodnotí její podmínu.

Níže máme dva obdobné příklady pro a while smyčky:

continue ve for loop

```
1 for cislo in range(20):
2     if cislo % 2 == 0:
3         continue
4     print(cislo)
```

```
1
3
5
...
17
19
```

continue ve while loop

```
1 # Pocitadlo
2 number = -1
3
4 # While Loop
5 while number < 20:
6     number += 1
7     if number % 2 == 0:
8         continue
9     print(number)
```

```
1
3
5
...
17
19
```

Smyčka **while** potřebuje explicitní úpravu procházené proměnné, kdežto smyčka **for** upravuje danou proměnnou místo nás. Smyčka for je také v tomto případě přehlednější.

Tvoje úloha

Zkus si tipnout, co by se stalo, kdybychom klíčové slovo **continue** nevložili pod podmínkový výraz?

1. continue a for loop

Kdybychom použili for loop, žádný kód po klíčovém slovu **continue** by se neprovedl. V našem případě by se nevytisklo žádné číslo. **Jakmile smyčka projde všechny prvky, ukončí se.**

```
1 # For Loop
2 for cislo in range(20):
3     continue
4     print(cislo)
```

Nic není vytištěno:

2. continue a while loop

Pokud je proměnná `cislo` změněna po klíčovém slovu `continue`, **smyčka se nikdy neukončí**. Proč? Podmínka `cislo < 20` nebude nikdy vyhodnocena jako False.

```
1 number = -1
2 while number < 20:
3     continue
4     number += 1
5     print(number)
```

Výsledkem bude nekonečná smyčka, kterou můžeme přerušit zkratkou **Ctrl+C**:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyboardInterrupt
```

Z této smyčky se dostaneme, ale nic nebude vytištěno.

```
1 number = -1
2 while number < 20:
3     number += 1
4     continue
5     print(number)
```

Nic nebude vytištěno:

Break podrobně

ONLINE PYTHON AKADEMIE / FOR CYKLUS / SMYČKA FOR / BREAK PODROBNĚ

Obdobně jako u `continue`, jakmile Python narazí na `break`, kód na stejně úrovni (se stejným počtem odsazení) nebude proveden. Výraz `break`, stejně jako `continue`, by měl být v podminkovém výrazu.

Na rozdíl od `continue`, po klíčovém slovu `break` kód nepokračuje hlavičkou smyčky, ale smyčka se ukončí a pokračuje v kódu pod smyčkou. Opět následují dva obdobné příklady pro for a while:

break ve for loop

```
1 for cislo in range(100):
2
3     if 40 < cislo < 60:
4         break
5     print(cislo)
6
7 print('KONEC')
```

```
0
1
2
...
39
40
KONEC
```

break ve while loop

```
1  cislo = 0
2  while cislo < 100:
3
4      if 40 < cislo < 60:
5          break
6      print(cislo)
7
8      cislo += 1
9
10 print('KONEC')
```

```
0
1
2
.
.
.
39
40
KONEC
```

V příkladu výše budou vytiskána jen čísla 0 a 40. Jakmile proměnná `cislo` bude rovna 41, podmínka `if 40 < cislo < 60:` se stane pravdivou, `break` se provede a skočí na `print('KONEC')`.

For Loop - else blok

ONLINE PYTHON AKADEMIE / FOR CYKLUS / SMYČKA FOR / FOR LOOP - ELSE BLOK

Smyčky for i while mohou mít navíc blok začínající klíčový slovem `else`. Indentace tohoto klíčového slova musí být stejná jako hlavička smyčky.

Kód uvnitř `else` bloku je proveden, pokud **blok smyčky (for/while) skončí bez přerušení - nenarází na break**.

else a for loop

Pokud písmena 'xz' nebyla nalezena, výraz `break` nebude nikdy proveden a bude vytisknuto - **Písmena "xz" nejsou uvnitř stringu**.

```
1 # String
2 muj_str = 'Python Loop'
3
4 # For Loop
5 for pismeno in muj_str:
6     if pismeno in 'xz':
7         break
8 else:
9     print('Písmena "xz" nejsou uvnitř stringu')
```

```
Písmena "xz" nejsou uvnitř stringu
```

Pokud narazíme na písmena 'x' nebo 'z' v našem stringu, vytiskneme - **Písmena nalezena**.

```
1 # String s 'x'
2 muj_str = 'Pzthon Loop'
3
4 # For Loop
5 for pismeno in muj_str:
6     if pismeno in 'xz':
7         print('Písmena nalezena')
8         break
9 else:
10    print('Písmena "xz" nejsou uvnitř stringu')
```

```
Písmena nalezena
```

else a while loop

else: blok funguje také se smyčkou **while**

```
1 # String
2 muj_str = 'Python Loop'
3
4 # While Loop
5 while muj_str:
6     if muj_str[0] in 'xz':
7         break
8     muj_str = muj_str[1:]
9 else:
10    print('Písmena "xz" nejsou uvnitř stringu')
```

Písmena "xz" nejsou uvnitř stringu

For Loop - enumerate()

ONLINE PYTHON AKADEMIE / FOR CYKLUS / SMYČKA FOR / FOR LOOP - ENUMERATE()

Nyní se naučíme používat novou funkci - **enumerate()**. Ta se nám bude hodit, když budeme chtít **projít indexy a prvky zároveň**.

Zatím jsme si ukázali, jak můžeme pomocí **range()** generovat indexy jednotlivých prvků v procházeném objektu. Následně jsme index použili, abychom manipulovali s prvky v objektu:

```
1 # String
2 nejaky_string = 'For loops podporují iterační protokol'
3
4 # For Loop
5 for index in range(len(nejaky_string)):
6     znak = nejaky_string[index]
7     if index % 2 == 0:
8         znak = znak.upper()
9
10    print(znak, end = '')
```

FoR LoOpS PoDpOrUjÍ ItErAčNí pRoToKoL

Nicméně používání této techniky není úplně 'pythonistické'. Preferované používání for loop je rovnou procházet prvky objektu, ne jeho indexy. Tímto způsobem můžeme ušetřit několik řádků kódu. A pokud potřebujeme manipulovat s daným prvkem pomocí indexu, můžeme použít built-in funkci **enumerate()**

```
1 # String
2 nejaky_string = 'For loops podporují iterační protokol'
3
4 # For Loop
5 for index, znak in enumerate(nejaky_string):
6     if index % 2 == 0:
7         znak = znak.upper()
8
9     print(znak, end = '')
```

FoR LoOpS PoDpOrUjÍ ItErAčNí pRoToKoL

Místo toho, abychom zjistili délku sekvence a potom generovali jinou sekvenci ve formě range objektu, používáme funkci enumerate, která vrátí 2členný tuple - (index_prvku, prvek).

```
>>> list(enumerate('For loops podporují iterační protokol'))
[(0, 'F'), (1, 'o'), (2, 'r'), (3, ' '), (4, 'l'), (5, 'o'), (6, 'o'), (7, 'p'), (8, 's'), (9, ' '),
  (10, ' ')]
```

Krásnější provedení kódu výše.

```
1 # String
2 nejaky_string = 'For loops podporují iterační protokol'
3
4 # For Loop
5 for index, znak in enumerate(nejaky_string):
6     print(index, "->", znak)
```

0 -> F

```
1 -> o
2 -> r
3 ->
4 -> l
5 -> o
...
31 -> o
32 -> t
33 -> o
34 -> k
35 -> o
36 -> l
```

Když procházíme enumerate objekty, v každém cyklu se vrátí jeden 2členný tuple. První prvek tuplu je číslo, které reprezentuje index prvku v procházeném objektu, druhý prvek je prvek samotný `enumerate()` vrácí speciální objekt enumerate.

```
>>> enumerate('abcde')
<class 'enumerate'><enumerate object at 0x7f6800e50ab0>
```

K PROCVIČENÍ

Úkol 21: Fizz Buzz

SPUSTIT ÚKOL

Populární programovací úkol pro cvičení cyklů a podmínek. Vytiskni číslo 1-100 a rozhodni, jestli jsou dělitelná 3, 5 nebo obojím.

⌚ 10 min. ⚡ střední

Úkol 22: Šachovnice

SPUSTIT ÚKOL

Chceš si zahrát šachy? Použij smyčku for a vytiskni si šachovnici!

⌚ 20 min. ⚡ pokročilé

Úkol 23: Převod času

SPUSTIT ÚKOL

Napiš program pro převod času z 24hodinovém formátu do anglického času.

⌚ 20 min. ⚡ těžší

Úkol 24: Skript na seřazení

SPUSTIT ÚKOL

Let's create a program that will sort any list of strings according to alphabetical order, using for loops and .pop() method.

⌚ 20 min. ⚡ střední

Úkol 25: String to list

SPUSTIT ÚKOL

Převeď string s čísly na seznam!

⌚ 10 min. ⚡ střední

Úkol 26: Samohlásky a souhlásky

SPUSTIT ÚKOL

Napiš skript pro spočítání počtu samohlásek a souhlásek ve větě.

⌚ 20 min. ⚡ těžší

Úkol 27: Dělitel

SPUSTIT ÚKOL

Write a script which asks user for inputs 'start', 'stop' and 'divisor', then prints a list of those numbers in range `start` - `stop`, that are divisible by `divisor`.

⌚ 20 min. ⚡ těžší

Úkol 28: Počty čísel

SPUSTIT ÚKOL

Ask user for a number and then determine, whether the number is odd or even. You should use ternary operator to solve this task.

⌚ 10 min. ⚡ těžší

KVÍZ

Range

ONLINE PYTHON AKADEMIE / FOR CYKLUS / KVÍZ / RANGE

1/11

Co je hlavní využití objektů range?

- _____
- A. Generování desetinných, nebo celých čísel v daném rozmezí.
 - B. Generování celých čísel v daném rozmezí.
 - C. Generování náhodných celých čísel v rozmezí od - do.
 - D. Generování náhodných desetinných čísel v rozmezí od - do.

For Loop

ONLINE PYTHON AKADEMIE / FOR CYKLUS / KVÍZ / FOR LOOP

1/4

Která z následujících hlaviček for loop je správné, když chceme projít string `'Python'` uložený v proměnné `my_str`?

- _____
- A. `for in pismeno my_str:`
 - B. `pismeno in for my_str:`
 - C. `for my_str in pismeno :`

D. for pismeno in my_str:

[DALŠÍ LEKCE](#)

>_ Terminál

