 Python akademie - lekce 6 - 21.11.2024


▼

06_00: Opakování po páté lekci!

▼ Ukázka #01: Použití walrus operátoru :=

▼ 1. Základní přiřazení hodnoty:

```
muj_str_1 = "bez_walruse"
print(muj_str_1)
```


 bez_walruse

- Proměnná `muj_str_1` dostane hodnotu "bez_walruse". Znak `=` zde znamená přiřazení, tj. ukládáme hodnotu do proměnné.
- Funkce `print()` vypíše hodnotu uloženou v proměnné `muj_str_1` na obrazovku.
- Výstup:

bez_walruse

▼ 2. Použití walrus operátoru – nesprávný zápis:

```
muj_str_2 := "s_walrusem"
print(muj_str_2)
```

 File "<ipython-input-22-771b8ba29ccc>", line 1
muj_str_2 := "s_walrusem"
 ^
SyntaxError: invalid syntax

- Tento kód obsahuje chybu, protože walrus operátor `:=`, který kombinuje přiřazení a vrácení hodnoty, vyžaduje kulaté závorky, pokud je použit mimo větší výrazy.
- Chyba, kterou Python vrátí:

SyntaxError: invalid syntax

▼ 3. Správné použití walrus operátoru:

```
(muj_str_3 := "s_walrusem")
print(muj_str_3)
```

 s_walrusem

- Zde je použití správné díky kulatým závorkám. Operátor `:=` přiřadí hodnotu "s_walrusem" do proměnné `muj_str_3` a zároveň vrátí tuto hodnotu, takže ji můžeme ihned použít v kódu.
- Funkce `print()` pak vypíše obsah proměnné `muj_str_3`.
- Výstup:

s_walrusem

Poznámka:

Walrus operátor se často používá pro efektivnější zápis, např. při práci se smyčkami nebo podmínkami.

Obecně pravidla pro používání a zapisování najdeme v [oficiální dokumentaci](#).

✓ Ukázka #02: Práce s tuple a přiřazení hodnot

✓ 1. Přiřazení hodnot jako tuple:

```
moje_cisla = 69, 96
print(moje_cisla)
```

```
(69, 96)
```

- Proměnná `moje_cisla` získá hodnotu `tuple`, což je typ datové struktury podobné seznamu, ale neměnitelné (immutable).
- Hodnota `(69, 96)` je tuple obsahující dvě čísla, která se vypíší pomocí funkce `print()`.
- Výstup:

```
(69, 96)
```

✓ 2. Použití walrus operátoru s tuple:

```
(moje_cisla := 69, 96)
print(moje_cisla)
```

```
69
```

- Tento kód přiřadí hodnotu pouze prvnímu číslu 69 díky tomu, že walrus operátor přidělí hodnotu prvnímu prvku tuple a druhý ignoruje.
- Výstup:

```
69
```

✓ 3. Rozdělení hodnot tuple na jednotlivé proměnné:

```
moje_cislo_1, moje_cislo_2 = 69, 96
print(moje_cislo_1)
print(moje_cislo_2)
```

```
69
96
```

- Python umožňuje přiřadit jednotlivé hodnoty tuple do více proměnných na jeden řádek.
- `moje_cislo_1` získá hodnotu 69 a `moje_cislo_2` hodnotu 96. Tyto hodnoty jsou pak postupně vypsány.
- Výstup:

```
69
```

```
96
```

✓ 4. Poznámka k chybnému použití přiřazení uvnitř tuple:

```
(moje_cislo_1, moje_cislo_2 = 69, 96)
print(f"{moje_cislo_1}; {moje_cislo_2}")
```

```
File "<ipython-input-7-ec8bb80555d3>", line 1
(moje_cislo_1, moje_cislo_2 = 69, 96)
                        ^
```

```
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

- Chyba: Tento zápis způsobí chybu, protože přiřazení = není povoleno uvnitř tuple. Python očekává použití walrus operátoru := nebo jiné validní syntaxe.
- Chybová zpráva:

```
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

✓ 5. Pokročilé použití tuple a walrus operátoru

```
(moje_cislo_1, moje_cislo_2 := 68, 86)
```

```
↔ (69, 68, 86)
```

```
moje_cislo_1
```

```
↔ 69
```

```
moje_cislo_2
```

```
↔ 68
```

► ? Vysvětlení ?

```
(moje_cisla := 69, 96) == ((moje_cisla := 69), 96)
```

```
↔ True
```

```
x = (moje_cisla := 696, 96)
```

```
type(x)
```

```
↔ tuple
```

```
x[1] is moje_cisla
```

```
↔ False
```

✓ Ukázka #03: Práce se seznamem (list) a smyčkami

```
muj_list = ["a", "b", "c", "d", "e", "f"]
```

✓ 1. Použití smyčky while:

```
while muj_list:
    udaj = muj_list.pop()
    print(udaj)
```


```
↔ f
    e
    d
    c
    b
    a
```

Co se děje:

- Seznam `muj_list` obsahuje hodnoty "a", "b", ..., "f".
- Smyčka `while` se opakuje, dokud seznam není prázdný.
- Metoda `pop()` odstraní poslední prvek seznamu a zároveň ho vrátí, aby mohl být zpracován.



✓ 2. Použití smyčky for:

```
for udaj in muj_list:
    print(udaj)
```



```
a
b
c
d
e
f
```


- Smyčka `for` projde každý prvek seznamu od začátku do konce. Na rozdíl od `while` smyčky nemění seznam, takže data zůstávají zachována.

►  Vysvětlení 

▼ Ukázka #04:Specifické chování `for` smyčky


▼ 1. Kód s modifikací proměnné uvnitř smyčky:

```
for cislo in range(0, 10):
    if cislo == 9:
        cislo = 0
    print(cislo)
```



```
0
1
2
3
4
5
6
7
8
0
```

- Smyčka `for` iteruje přes čísla v rozsahu od 0 do 9 (funkce `range()` generuje tato čísla).
- Pokud je hodnota `cislo` rovna 9, nastaví se na 0. Toto nastavení ovlivní pouze aktuální hodnotu uvnitř smyčky, ne však celkový rozsah iterace.
- Smyčka `for` iteruje přes hodnoty předem vytvořeného seznamu (`range(0, 10)`) a nemění tento seznam ani jeho hodnoty. Jakmile se přesune na další iteraci, pokračuje podle původních hodnot.

►  Vysvětlení 
