Proměnné, fukce a typy

= není “rovná se”

```
a = 123
```

= reprezentuje přiřazení, kde a je proměnná

```
123 = a
```

Abstrakce pomocí proměnných

```
draha = 300
cas = 20

rychlost = draha / cas
```

Proměnnou můžeme použít místo hodnoty

```
print(rychlost)
```

Existují různé typy objektů a nechovají se stejně

Funkce *type* vypíše typ objektu

```
type(12)
```

```
int
```

```
a = "text"
type(a)
```

```
str
```

```
cele_cislo = 12
text = "dvanact"
kvazi_realne_cislo = 1/3
seznam = [1, "dva", 3.0]
```

Operátory v Pythonu

+	-	*	**	/	//	%	@
<	>	<=	>=	==	!=		
<<	>>	&	|	^	~	:=	

Různé typy se chovají různě

Vyzkoušejme sčítání pro tyto věci:

```
print(cele_cislo + cele_cislo)
```

```
print(cele_cislo + text)
```

```
print(cele_cislo + kvazi_realne_cislo)
```

```
print(cele_cislo + seznam)
```

```
print(text + text)
```

```
print(seznam + seznam)
```

Změna typu je občas možná

```
a = 12 / 5
print(a)
type(a)
```

```
b = int(a)
print(b)
type(b)
```

```
c = str(b)
print(c)
type(c)
```

Někdy je změna implicitní:

```
d=10
type(d)
```

```
d = d + 0.1
print(d)
type(d)
```

Cvičení 1.

Zjistěte jaký typ mají následující objekty:

```
a = 1.32
b = 1,32
c = [1, 32]
d = (1, 32)
e = "132"
```

Cvičení 2.

Ověřte jak se chovají operátory +, -, *, / pro dvojce těchto typů:

- ➔ str a str
- ➔ list a list
- ➔ int a list
- ➔ int a str
- ➔ str a int

Nápověda:

- ➔ str je text (string), např. “text”
- ➔ list je seznam, např. [1, 2 , 3]
- ➔ int je celé číslo (integer), např. 12

Rekapitulace

Zatím víme, že:

- ➔ Proměnná (a, cele_cislo, …)
- ➔ Hodnota (12, “text”, …)
- ➔ Operátory (+, -, …)
- ➔ Příkazy (a = 12)

Co jsou ale print(), type(), int(), … čili věci se závorkami?

Funkce -> více abstrakce

```
def vypis_dvakrat(x):
    print(x)
    print(x)
```

```
vypis_dvakrat("neco")
vypis_dvakrat(12)
```

Nezapomeňte na : a intendovaný blok jinak dostanete SyntaxError.

To co do funkce dáváme (x v definici) se jmenuje argument (parametr) funkce.

Funkce s více argumenty a vrácením hodnoty

```
def vypis_a_secti(a, b):
    print(a, "+", b)
    return a + b
```

```
vysledek = vypis_a_secti(1, 3)
print(vysledek)
```

```
print(vypis_a_secti(1, 3))
```

```
vypis_a_secti(1, 3)
```

Cvičení 3.

Napište funkci která:

- ➔ Vypíše argument a jeho typ
- ➔ Přičte jedničku k argumentu a vrátí jeho hodnotu

Funkce bez vrácení hodnoty

Když vynecháte return, je to stejné jako kdybyste napsali return None:

```
def vypis_dvakrat(x):
    print(x)
    print(x)
```

```
vysledek = vypis_dvakrat(12)
```

```
print(vysledek)
```

```
type(vysledek)
```

Co je to None

```
type(None)
```

None je definované jako nulová hodnota, nebo žádná hodnota. None je stejný jako 0, False nebo prázdný string. None je svůj vlastný typ (NoneType) a jenom None může být None

Kompozice

Funkce je možné skládat:

```
print(type(vypis_dvakrat(12)))
```

Parametry funkce jsou lokální proměnné

```
def plus_2(a):
    c = a + 2
    return c

a = 1
b = plus_2(3)

print(b)
print(a)
print(c)
```

Parameter a ve funkci zakryje globální proměnnou a.

Parametry funkce jsou lokální proměnné, ale …

```
def a_plus_2():
    c = a + 2
    return c

a = 1
b = a_plus_2()
print(b)
```

Parametry funkce jsou lokální proměnné, ale …

Toto neudělá co si nejspíš myslíte:

```
def a_is_a_plus_2(a):
    a = a + 2
    print(a)

a = 1
a_is_a_plus_2(a)
print(a)
```

```
def a_is_a_plus_2():
    a = a + 2

a = 1
a_is_a_plus_2()
print(a)
```

Je funkce objekt? (Můžeme ji přiřadit do proměnné?)

```
def plus_2(a):
    c = a + 2
    return c

fun = plus_2
fun(3)
```

Pozor na závorky!

```
fun = plus_2(3)   # int
fun = plus_2      # funkce
```

Funkce jako argument funkce

```
def udelej_neco_s_cislem_tri(fun):
    return fun(3)

udelej_neco_s_cislem_tri(vypis_dvakrat)
```

```
def print_type(x):
    print(type(x))

udelej_neco_s_cislem_tri(print_type)
```

Cvičení 4.

- ➔ Vytvořte funkci, která vezme jeden argument - jinou funkci a zavolá ji dvakrát
- ➔ Otestujte vaši funkci

Metoda - funkce trochu jinak

V objektově-orientovaných jazycích jsou funkce často přímo napojené na objekty.

```
text = "stop"

print(text.upper())
```

```
def upper(string):
    ...
    return upper_string
```

Metoda - funkce trochu jinak

V objektově-orientovaných jazycích jsou funkce často přímo napojené na objekty.

```
text = "stop"

print(text.upper())
```

- ➔ upper je metoda pro str
- ➔ obj.x je syntax který znamená “*x v obj*”
- ➔ metodu můžeme chápat jako funkci, která dostane jako první argument objekt, ze kterého je volána

```
def upper(string):
    ...
    return upper_string
```

Nebudeme si všechno programovat -> import

```
import math

print(math.sin(3.14/2))
```

```
import math as m

print(m.sin(3.14/2))
```

```
from math import sin

print(sin(3.14/2))
```

```
from math import *

print(sin(3.14/2))
print(cos(3.14/2))
```

Cvičení 5.

Napište funkci, která přepočítá úhel ve stupních na radiány a funkci inverzní. Číslo π najdete v knihovně math:

```
import math
print(math.pi)
```

Cvičení 6.

Napište funkci, která použije 1 argument a spočítá obvod a obsah kruhu, kde zadaný argument poslouží jako poloměr:

```
import math
print(math.pi)
```

```
def polomer_a_obsah(r):
    ...
```