

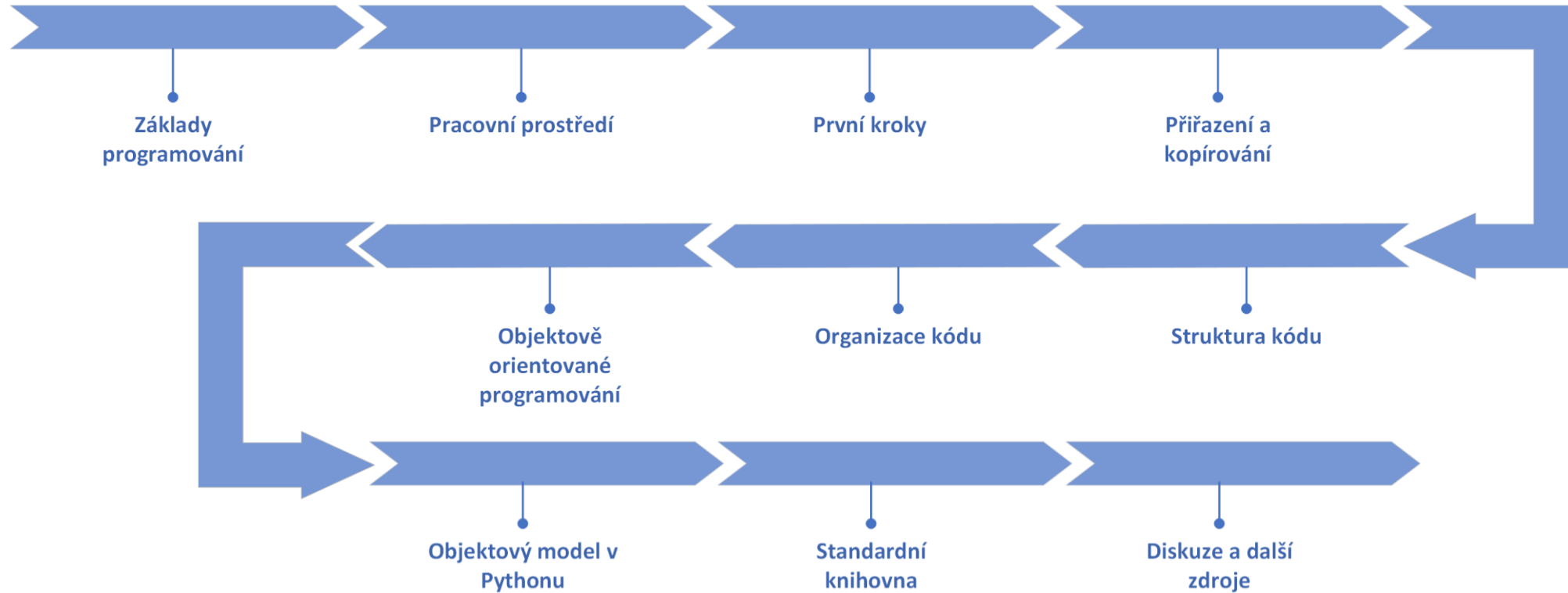
PYTHON - ZÁKLADY PROGRAMOVÁNÍ

(PYTH1)

ictPRO

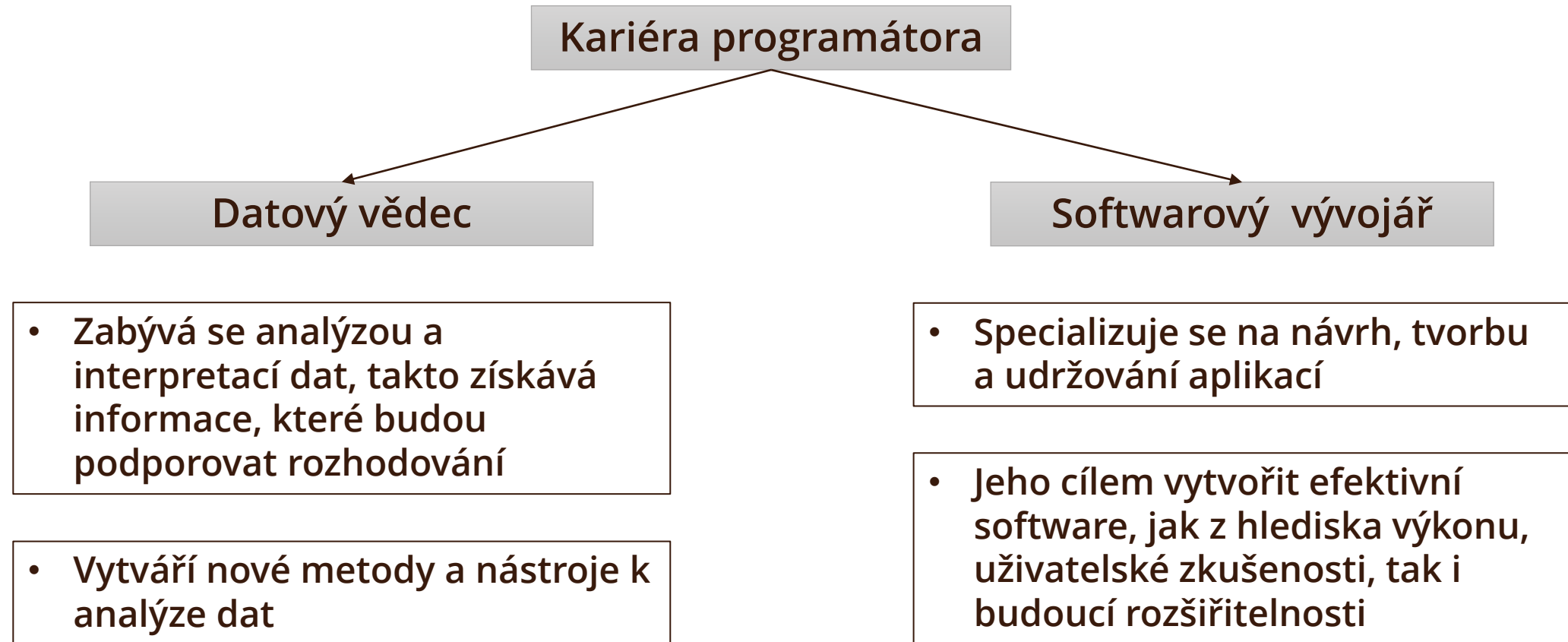


NÁPLŇ KURZU



ZÁKLADY PROGRAMOVÁNÍ

PRÁCE S DATY A VÝVOJ SOFTWARE



Některé nástroje a techniky se mohou shodovat, rozdílný je především cíl

ZÁKLADY PROGRAMOVÁNÍ

VÝBĚR PROGRAMOVACÍHO JAZYKA

- Znalosti v týmu, šance získat nové vývojáře či křivka učení

- Velikost komunity, knihovny a vývojové nástroje

- Statické x dynamické typování

- Silné x slabé typování

- Výkon a škálovatelnost

ZÁKLADY PROGRAMOVÁNÍ

SILNÉ A SLABÉ STRÁNKY PYTHONU

- Znalosti v týmu, šance získat nové vývojáře či křivka učení - Python je nejrozšířenější programovací jazyk, křivka učení je strmá (příznivá)
- Velikost komunity, knihovny a vývojové nástroje - Rozsáhlé komunity, množství knihoven, IDE, jak pro vývoj software, tak pro datovou analýzu
- Statické x dynamické typování – Python je dynamicky typovaný, cenou za flexibilitu a jednodušší zápis kódu je menší kontrola při překladu
- Silné x slabé typování – Python je silně typovaný, je nutné explicitně přetypovávat, výhodou je lepší kontrola při překladu
- Výkon a škálovatelnost – dynamické a interpretované jazyky jako Python jsou pomalejší, ale jeho knihovny bývají tvořené v rychlejších jazycích

Python je díky možnosti rychlého vývoje a množství knihoven velmi silný nástroj v práci s daty a ve strojovém učení je přímo číslo jedna.

PRACOVNÍ PROSTŘEDÍ

INSTALACE PYTHONU

- Poslední verze 3.12 python.org/downloads/

Active Python Releases

For more information visit the [Python Developer's Guide](#).

| Python version | Maintenance status | First released | End of support | Release schedule |
|----------------|----------------------------|----------------------|----------------|-------------------------|
| 3.13 | prerelease | 2024-10-01 (planned) | 2029-10 | PEP 719 |
| 3.12 | bugfix | 2023-10-02 | 2028-10 | PEP 693 |
| 3.11 | bugfix | 2022-10-24 | 2027-10 | PEP 664 |
| 3.10 | security | 2021-10-04 | 2026-10 | PEP 619 |
| 3.9 | security | 2020-10-05 | 2025-10 | PEP 596 |
| 3.8 | security | 2019-10-14 | 2024-10 | PEP 569 |

```
C:\Users\PetrF>python --version
Python 3.12.0
```

PRACOVNÍ PROSTŘEDÍ

PRÁCE V PŘÍKAZOVÉM ŘÁDKU

```
C:\Users\PetrF>python
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello Word")_
```

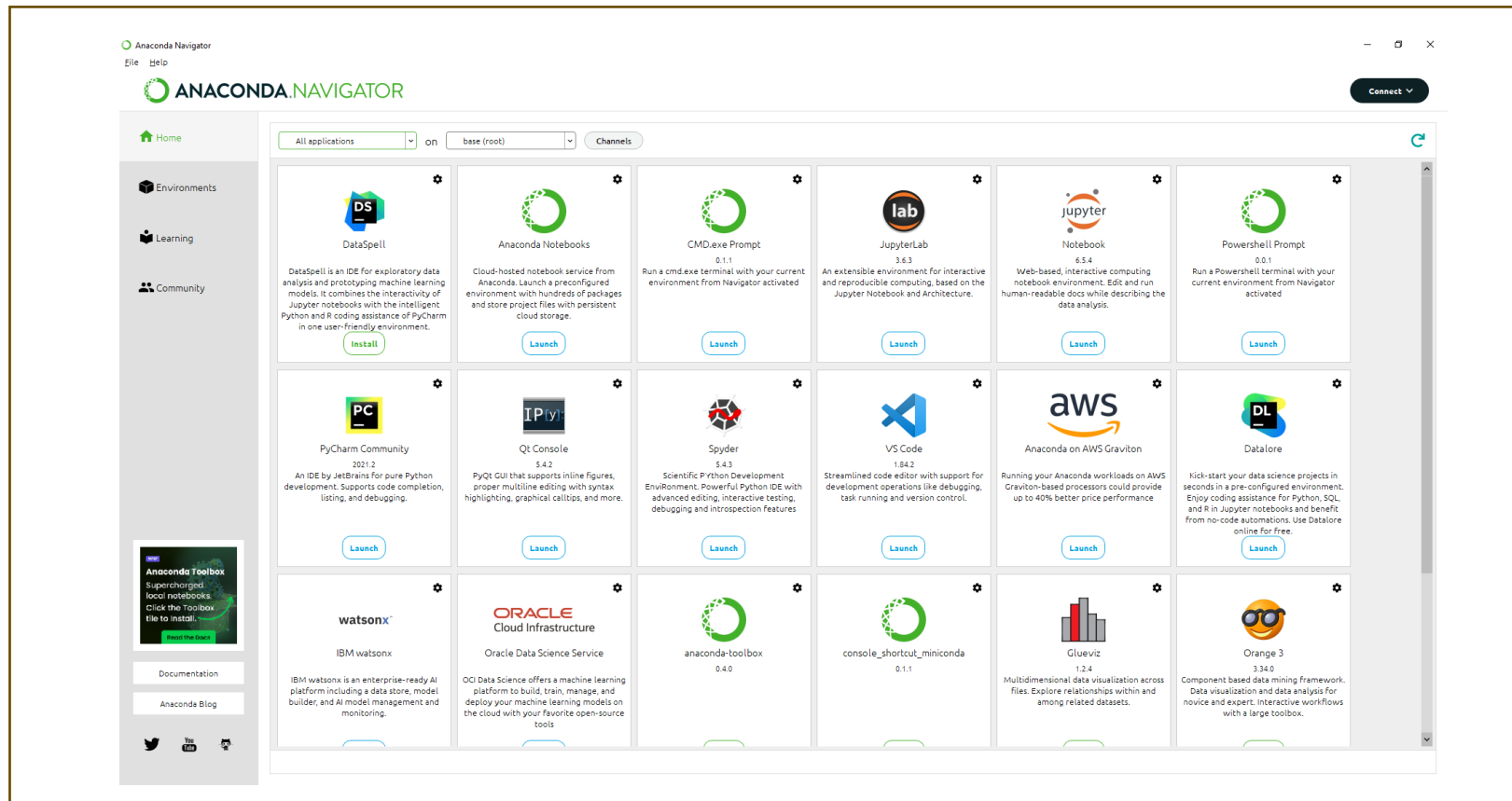
```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello Word")
Hello Word
>>>
```

PRVNÍ KROKY

ÚPRAVA PROGRAMŮ V TEXTOVÉM EDITORU

Datová analýza

- vědecká platforma Anaconda <https://www.anaconda.com/download>

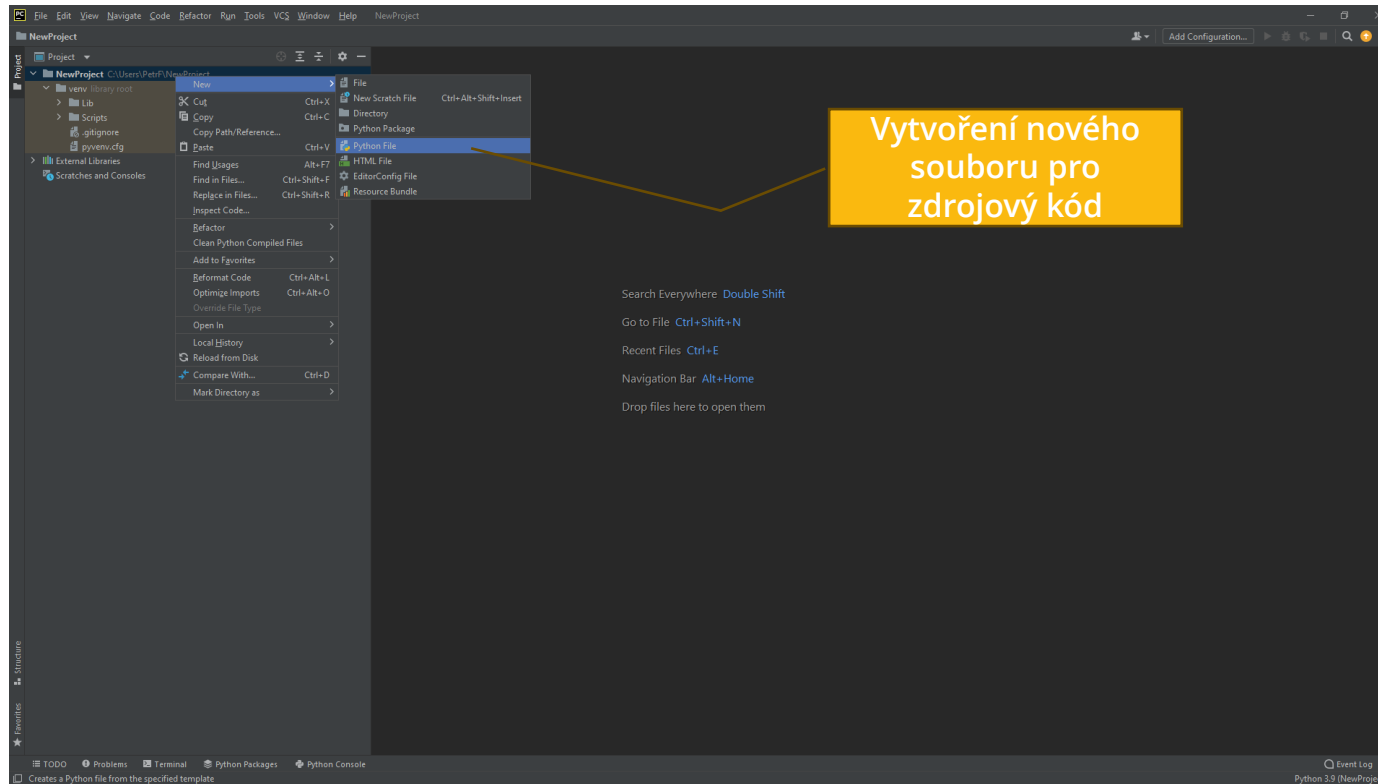


PRVNÍ KROKY

ÚPRAVA PROGRAMŮ V TEXTOVÉM EDITORU

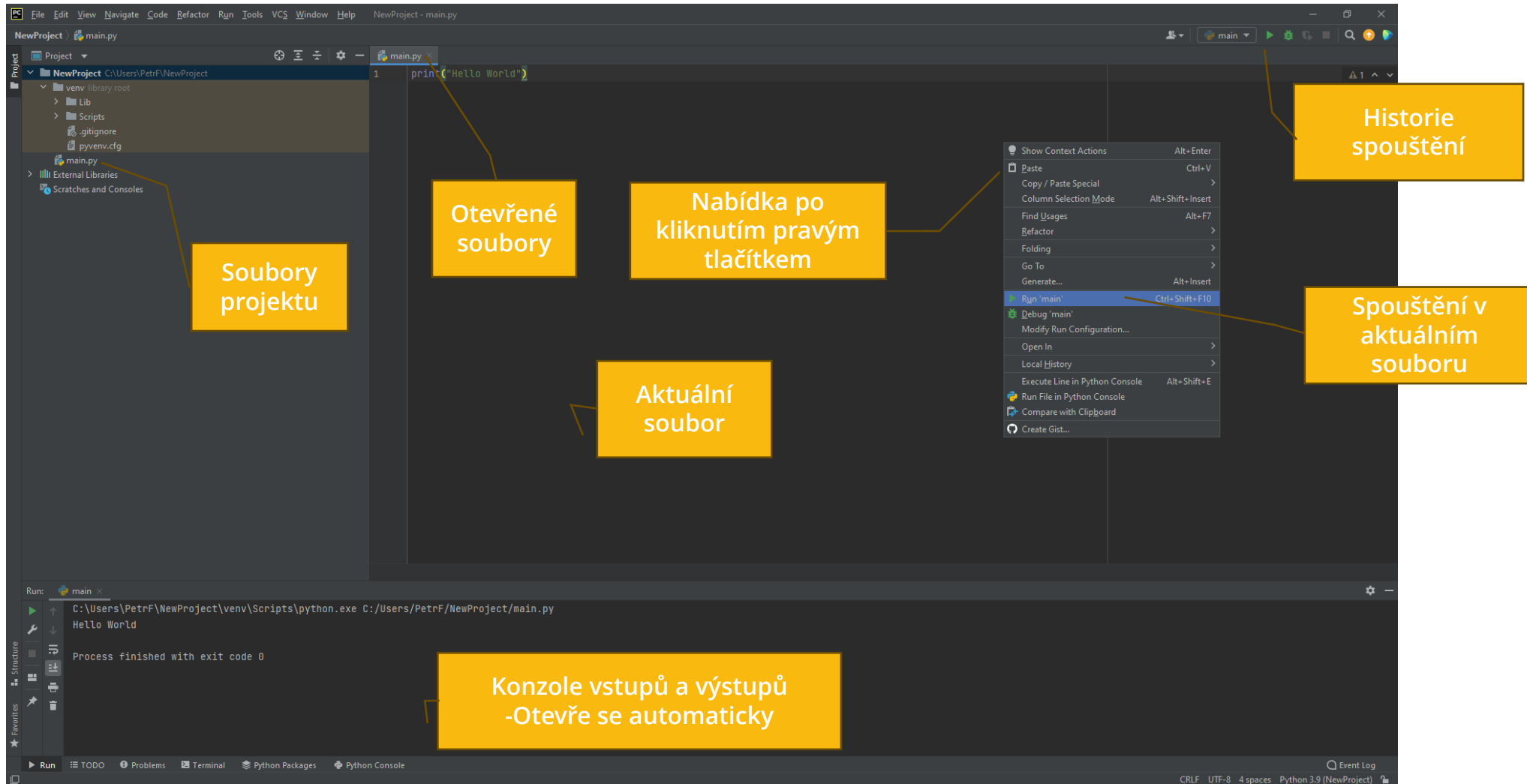
Softwarový vývoj

- IDE Pycharm <https://www.jetbrains.com/pycharm/download>



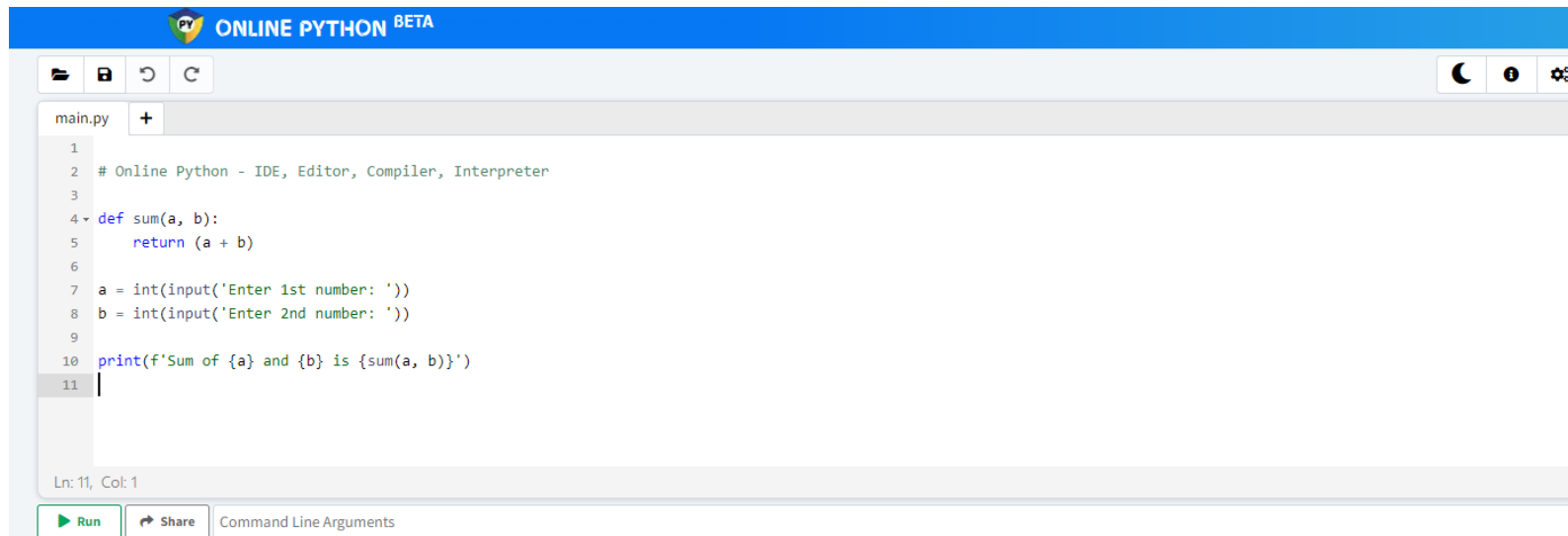
PRVNÍ KROKY

ÚPRAVA PROGRAMŮ V TEXTOVÉM EDITORU



PRVNÍ KROKY ONLINE NÁSTROJE

- Existují i online editory i s kompilátorem a interpretrem
- Například <https://www.online-python.com/>
- Výhoda je, že kód je možné sdílet pomocí URL odkazu



The screenshot displays the 'ONLINE PYTHON BETA' web interface. At the top, there's a blue header with the logo and title. Below it is a toolbar with icons for file operations (new, open, save, undo, redo) and settings (theme, help, settings). The main area is a code editor with a file named 'main.py'. The code is a Python script that defines a 'sum' function and takes two user inputs to calculate their sum. The code is as follows:

```
1
2 # Online Python - IDE, Editor, Compiler, Interpreter
3
4 def sum(a, b):
5     return (a + b)
6
7 a = int(input('Enter 1st number: '))
8 b = int(input('Enter 2nd number: '))
9
10 print(f'Sum of {a} and {b} is {sum(a, b)}')
11
```

At the bottom, there's a status bar showing 'Ln: 11, Col: 1' and buttons for 'Run' and 'Share', followed by a text input for 'Command Line Arguments'.

PRVNÍ KROKY

PROMĚNNÉ A HODNOTY

- Proměnná je vytvořena (deklarována), když do ní poprvé uložíme hodnotu
- Datový typ proměnné bude určen automaticky právě na základě této proměnné (dynamické typování)

```
cislo = 108
```

- Obsah proměnné nebo hodnotu můžeme zobrazit pomocí funkce print()

```
print(cislo)
```

- Nejprve je vyhodnocen výraz na pravé straně za rovnítkem. Až výsledná hodnota je přiřazena do proměnné a ta také určí datový typ proměnné

```
vysledek = 54 + 54
```

```
vysledek1 = vysledek + cislo
```

- Datový typ proměnné můžeme zjistit pomocí funkce type()

```
print(type(vysledek1))
```

PRVNÍ KROKY

POČÍTÁNÍ A PRÁCE S ČÍSLY

Číselné datové typy a základní operace

- Celá čísla: datový typ int

```
cislo = 108
```

- Desetinná čísla: datový typ float

```
desetinne_cislo = 5.2
```

- Sčítání (+), odčítání (-) a násobení (*) dvou celých čísel se vytvoří celé číslo. Pokud jedno z čísel je desetinné číslo, vznikne desetinné číslo.
- Při dělení (/) vznikne vždy desetinné číslo
- Při celočíselném dělení (//) dvou celých čísel je výsledkem celé číslo

- Operátor modulo (%) vrací zbytek po celočíselném dělení
- Pokud jej aplikujeme na dvě celá čísla, výsledkem je celé číslo. Pokud jedno z čísel je desetinné číslo výsledkem je desetinné číslo.

PRVNÍ KROKY

PRÁCE S TEXTEM

Textový datový typ a přetypování

- Text je reprezentován řetězcem znaků (stringem): datový typ `str`

```
text = "Hello Word"  nebo  text = 'Hello Word'
```

- Pokud chceme spojit textovou hodnotu například s číselnou hodnotou, musíme nejprve explicitně číselnou hodnotu přetypovat na textovou

```
cislo = 108  
slozeny_text = text + str(cislo)
```

Jelikož Python je silně typovaný, obecně pokud potřebujeme hodnotu určitého datového typu použít jako jiný datový typ musíme jí explicitně přetypovat pomocí funkce (např. `str()`, `int()`, `list()`...)

PRVNÍ KROKY

PRÁCE S TEXTEM

Vybrané metody pro práci s textem

- Převedení znaků v řetězci na malá/velká písmena

```
text = "Hello Word"  
text_mala_pismena = text.lower()  
text_velka_pismena = text.upper()
```

Tyto metody jsou volané
přímo na objektu textu

- Získání části textu na základě zadání indexů znaků, tzv. výřez (slicing)

```
cast_textu = text[6:8] cast_textu = text[6:] cast_textu = text[:5]
```

- Zjištění počtu znaků v řetězci

```
delka = len(text)
```

Není včetně

Po konec

Od začátku

- Získání znaku na základě zadání jeho indexu

```
znak = text[6]
```

PRVNÍ KROKY

KOMUNIKACE S UŽIVATELEM

- Pomocí funkce `input()` získáme vstup uživatele
 - Tato funkce zároveň může zobrazit text s výzvou (prompt)
 - Tato funkce vstup uživatele získá vždy v datovém typu `str`, i když uživatel zadá číslice
- ```
vstup = input("zadej text: ")
```

- Pokud očekáváme číselné hodnoty, nabízí se přímo výstup funkce `input()` přetypovat

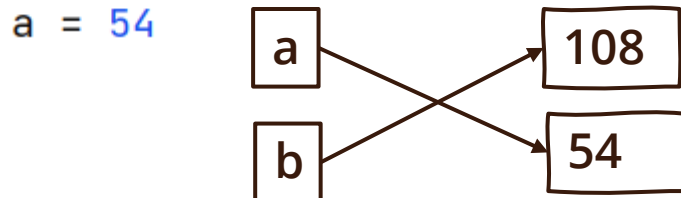
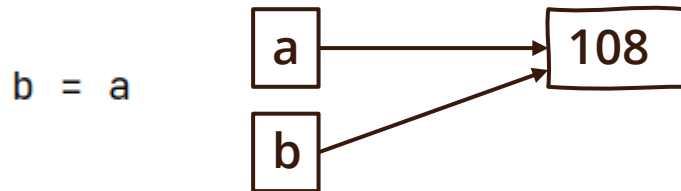
```
vstup = int(input("zadej cele cislo: "))
```

```
vstup = float(input("zadej desetinne cislo: "))
```

# PŘÍŘAZENÍ A KOPÍROVÁNÍ

## OBJEKTY A REFERENCE

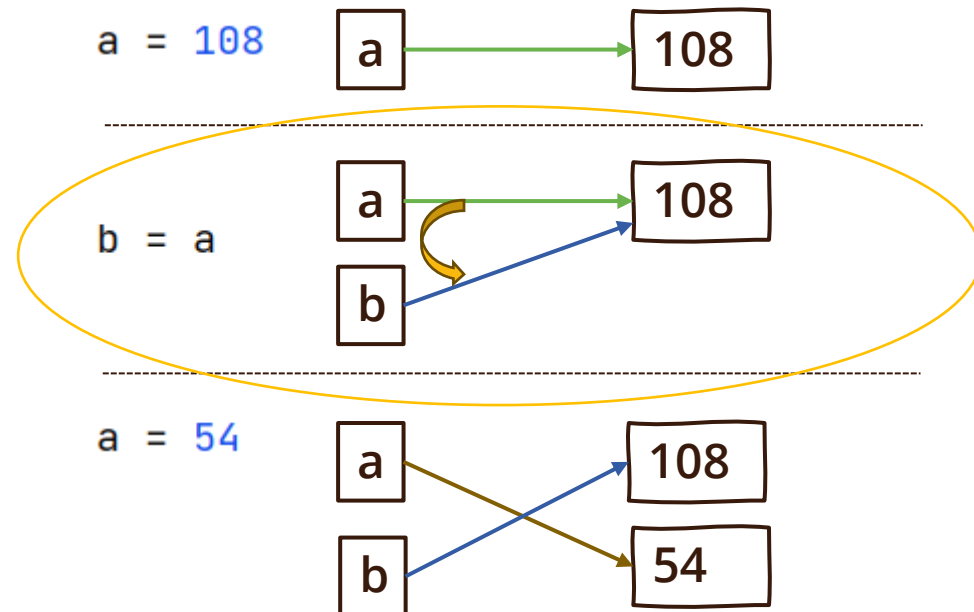
- Všechny hodnoty v Pythonu jsou objekty, v proměnných je pouze uložena reference(odkaz) na tyto objekty
- Číselné a textové hodnoty jsou neměnné (immutable). Proto princip referencí při jejich používání není tolik patrný. U objektu, které bude možné měnit, bude tento princip důležitý.



# PŘÍŘAZENÍ A KOPÍROVÁNÍ

## VÝZNAM OPERÁTORU PŘÍŘAZENÍ

- Operátor přiřazení pouze kopíruje reference, pokud na pravé straně je pouze jiná proměnná
- Zároveň pokud proměnná na levé straně ještě neexistuje, tak je vytvořena



# PŘÍŘAZENÍ A KOPÍROVÁNÍ

## VÝZNAM OPERÁTORU PŘÍŘAZENÍ

- Pokud je na pravé straně výraz, tak bude prvně vyhodnocen, vytvoří se v paměti konkrétní objekt a reference na něj bude opět uložena do proměnné
- Zároveň opět pokud proměnná na levé straně ještě neexistuje, tak je vytvořena



# PŘIŘAZENÍ A KOPÍROVÁNÍ

## ULOŽENÍ GLOBÁLNÍCH PROMĚNNÝCH

- Proměnné které vytvoříme mimo funkce, jsou globální
- Tyto proměnné jsou viditelné z celého programu

90

a = 108

Proměnná není v žádné funkci,  
můžeme jí použít kdekoli v programu

- Přehled všech globálních proměnných můžeme získat pomocí funkce globals()

```
print(globals())
```



# PŘÍŘAZENÍ A KOPÍROVÁNÍ

## SEZNAMY OBJEKTŮ

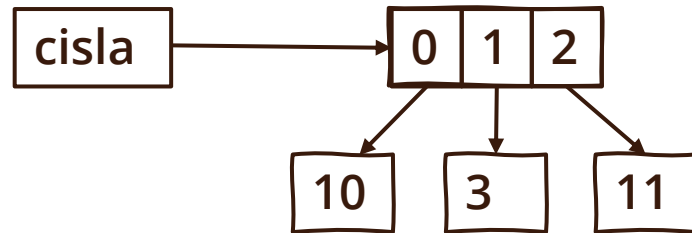
- K uložení více objektů do jedné proměnné použijeme seznam (datový typ list)
- Seznam můžeme vytvořit prázdný

```
cisla = []
```

- Seznam můžeme vytvořit rovnou i s objekty

```
cisla = [10, 3, 11]
```

- Každý objekt je v listu uložen pod unikátním indexem

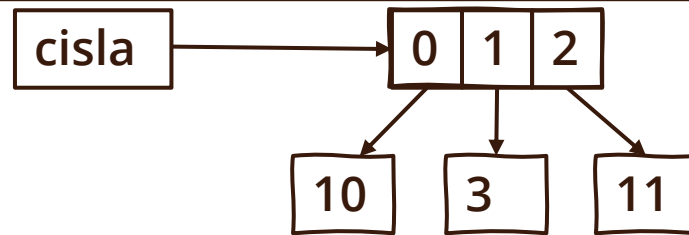


# PŘÍŘAZENÍ A KOPÍROVÁNÍ

## SEZNAMY OBJEKTŮ

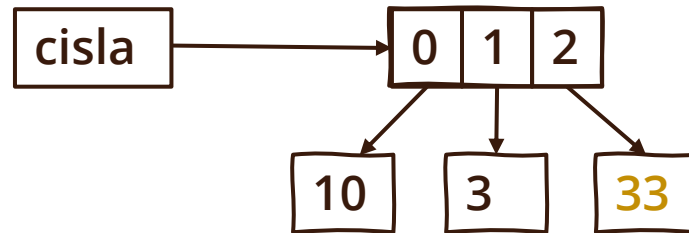
- Čtení ze seznamu pomocí indexu

```
cislo = cisla[1]
```



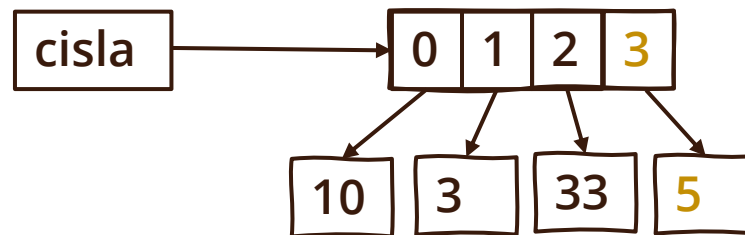
- Zápis do seznamu pomocí indexu

```
cisla[2] = 33
```



- Přidání objektu do listu pomocí metody append()

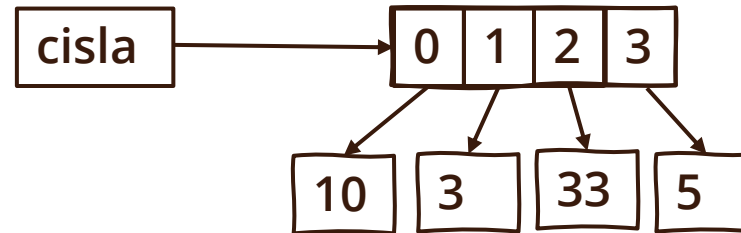
```
cisla.append(5)
```



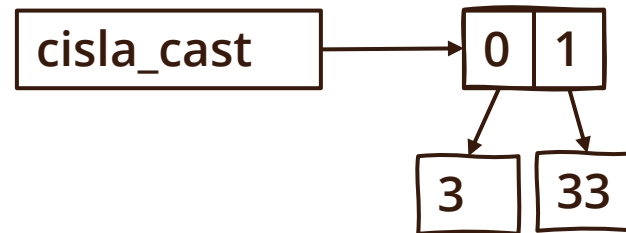
# PŘÍŘAZENÍ A KOPÍROVÁNÍ

## SEZNAMY OBJEKTŮ

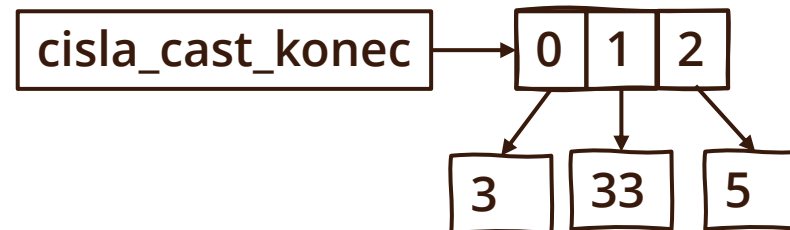
- Získání části seznamu na základě zadání indexů (pomoci výřezu)



```
cisla_cast = cisla[1:3]
```



```
cisla_cast_konec = cisla[1:]
```

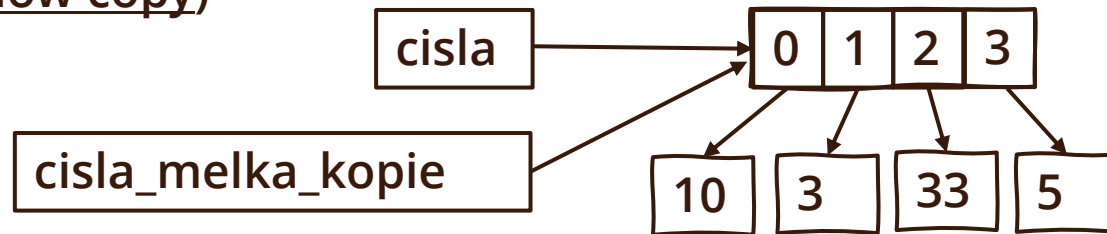


# PŘÍŘAZENÍ A KOPÍROVÁNÍ

## KOPÍROVÁNÍ OBJEKTŮ

- Pokud pouze zkopírujeme referenci na objekt, žádný nový objekt nevzniká, mluvíme tak o mělké kopii (shallow copy)

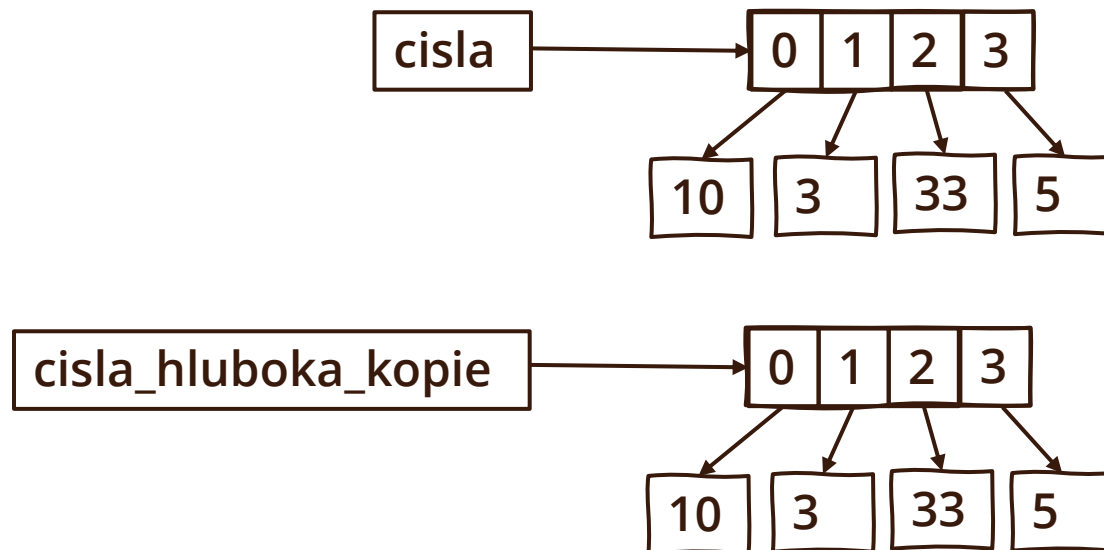
```
cisla_melka_kopie = cisla
```



- Pokud při kopírování vznikne nový objekt mluvíme o hluboké kopii (deep copy)

```
cisla_hluboka_kopie = cisla[:]
```

Hlubokou kopii můžeme vytvořit pomocí výřezu od začátku až po konec kopírovaného seznamu



# STRUKTURA KÓDU

## PŘÍKAZY A ODSAZENÍ

- Odsazení v Pythonu neslouží pouze k zpřehlednění kódu, ale vytváří skupiny příkazů, které budou vykonány v bloku (společně)
- Velikost odsazení je možné zvolit (např. několik mezer nebo tabulátor), je však třeba zvolené odsazení dodržovat
- Pokud chceme vytvořit nový blok příkazů navýšíme celkové odsazení řádku o zvolenou velikost
- Pokud chceme blok ukončit snížíme celkové odsazení řádku o tuto zvolenou velikost

# STRUKTURA KÓDU

## PODMÍNKY A CYKLY

- Podmínky  
if, elif, else

```
if cislo > 5:
 print(str(cislo) + " je vetsi nez 5")
elif cislo < 5:
 print(str(cislo) + " je mensi nez 5")
else:
 print(str(cislo) + " je rovno 5")
```

- Blok příkazů pod if se provede pouze pokud:
  - je logický výraz za if pravdivý
- Blok příkazů pod elif se provede pouze pokud:
  - je logický výraz za elif pravdivý a žádný přechodí logický výraz v podmínce nebyl pravdivý
- Blok příkazů pod else se provede pouze pokud:
  - žádný přechodí logický výraz v podmínce nebyl pravdivý



# STRUKTURA KÓDU

## PODMÍNKY A CYKLY

- Cykly  
for

```
cisla = [10, 3, 11]
soucet = 0
for cislo in cisla:
 soucet += cislo
print("Součet čísel je: " + str(soucet))
```

- Cyklus for se používá pro procházení prvků sekvence (např. seznam)
- Za klíčovým slovem for vytvoříme proměnnou, v které se prostřídají všechny prvky sekvence
- Pro každý prvek se blok příkazů pod klíčovým slovem for zopakuje

# STRUKTURA KÓDU

## PODMÍNKY A CYKLY

- Cykly  
for

```
text = "Hello World"
text_pozpatku = ""
for znak in text:
 text_pozpatku = znak + text_pozpatku
print("Text pozpatku: " + text_pozpatku)
```


- Sekvencí může být i textová hodnota
- Text projdeme znak po znaku

# STRUKTURA KÓDU

## PODMÍNKY A CYKLY

- Cykly  
for

```
for i in range(3):
 print("Hello World")
```



Zde konkrétně sekvence  
čísel: 0, 1, 2

- Pokud nemáme zájem procházet žádnou určitou sekvencí, ale chceme určité opakování bloku pod klíčovým slovem `for`, můžeme pomoci funkci `range()` vygenerovat sekvenci čísel

# STRUKTURA KÓDU

## PODMÍNKY A CYKLY

- Cykly  
for

```
cisla = [10, 3, 11]
cisla1 = [5, -5, 2]
sectena_cisla = []
for i in range(len(cisla)):
 sectena_cisla.append(cisla[i] + cisla1[i])
print("Seznam sectenych cisel: " + str(sectena_cisla))
```

- Někdy se hodí projít sekvence pomocí indexu
- Indexy je možné vygenerovat pomocí funkce range()

# STRUKTURA KÓDU

## PODMÍNKY A CYKLY

- Cykly  
while

```
vstup = ""
slov_napsano = 0
while vstup != "konec":
 vstup = input("zadej slovo: ")
 slov_napsano += 1
print("Celkove bylo zadano : " + str(slov_napsano) + " slov.")
```

- Blok příkazů pod klíčovým slovem nebo while se opakuje, dokud je logický výraz za ním pravdivý  
Cyklus while se hodí použít, když není známo kolikrát se průchod cyklem bude opakovat

# STRUKTURA KÓDU

## OŠETŘENÍ CHYB

```
delenec = 10
delitel = 0
try:
 vysledek = delenec / delitel
 print(vysledek)
except ZeroDivisionError:
 print("Nelze delit nulou")
```

- Blok příkazů, kde očekáváme, že se může chyba vyskytnout dáme pod klíčové slovo try
- Pod blok dáme společně s klíčovým slovem except typ očekávané chyby a pod toto klíčové slovo vytvoříme blok příkazů, kterou budou vykonány v případě, že chyba nastane



# STRUKTURA KÓDU

## OŠETŘENÍ CHYB

```
while True:
 try:
 cele_cislo = int(input("zadej cele cislo: "))
 print("Bylo zadano toto cele cislo: " + str(cele_cislo))
 break
 except ValueError:
 print("Nebylo zadano cele cislo.")
```

- Chybám můžeme předejít použitím podmínek a nemusí být tedy nutné je řešit pomocí ošetření chyb. Ale například k ošetření nevhodného vstupu uživatele, se použití konstrukce try/except nabízí.

# ORGANIZACE KÓDU

## PSANÍ FUNKCÍ

- Funkce je blok příkazů, který se provede, pokud je funkce zavolána
- Definování funkce začíná klíčovým slovem def, následuje volitelný název funkce, kterým bude funkce volána a kulaté závorky.
- Pokud funkce obsahuje parametry budou vyjmenované v kulatých závorkách

```
def vypis_trikrat_vetu():
 for i in range(3):
 print("Hello World")
```

```
vypis_trikrat_vetu()
vypis_trikrat_vetu()
```

# ORGANIZACE KÓDU

## PŘEDÁVÁNÍ PARAMETRŮ

- Parametry jsou lokální proměnné, do kterých bude při volání funkce dosazena hodnota
- Tato hodnota je kopie reference na objekt
- Pokud je tento objekt neměnný (např. str, int, float), nemůže funkce objekt změnit
- Změnitelné objekty jako seznam (list) může funkce upravit

```
def pozdrav(jmeno):
 print("Ahoj " + jmeno)
```

```
pozdrav("Pepa")
pozdrav("Jirka")
```

Objekt typu str,  
na který vede  
tato reference  
je neměnný

Nové číslo bude  
přidáno do seznamu,  
který je uložen pod  
globální proměnnou  
„moje\_cisla“

```
def pridej_cislo_do_seznamu(cisla, cislo):
 cisla.append(cislo)
```

```
moje_cisla = [1, 3, -5]
pridej_cislo_do_seznamu(moje_cisla, 8)
```

# ORGANIZACE KÓDU

## LOKÁLNÍ PROMĚNNÉ

- Lokální proměnné jsou viditelné jenom uvnitř funkce, ve které jsou deklarovány
- Pokud existuje globální proměnná se stejným jménem mimo funkci, je lokální proměnnou zastíněná (platí lokální proměnná)

```
def secti(scitanec, scitanec1):
 soucet = scitanec + scitanec1
 return soucet
```

```
print(secti(4, 7))
```

„scitanec“, „scitanec1“ a  
„vysledek“ jsou lokální  
proměnné

- Přehled všech lokálních proměnných můžeme získat pomocí funkce locals()

```
print(locals())
```

# ORGANIZACE KÓDU

## VRÁCENÍ HODNOTY

- Funkce může vracet hodnotu (v místě kde je funkce zavolána bude hodnota získána)
- Výraz, který je za klíčovým slovem return bude vyhodnocen a výsledná hodnota vrácena

```
def secti(scitanec, scitanec1):
 soucet = scitanec + scitanec1
 return soucet
```

```
print(secti(4, 7))
```

Výsledek nemusí být uložen do lokální proměnné. Může být vypočítán přímo za klíčovým slovem return.

```
def secti(scitanec, scitanec1):
 return scitanec + scitanec1
```

# ORGANIZACE KÓDU

## LOKÁLNÍ A GLOBÁLNÍ PROMĚNNÉ

- Pokud je to možné, je dobré se používání globálních proměnných vyhýbat, kód se stává nepřehledným a zvyšuje se riziko chyb

```
cisla = [0, 0, 0]
cisla1 = [0, 0, 0]
```

Globální proměnné

```
def pridej_cislo_do_seznamu(cisla, cislo):
 cisla.append(cislo)
 cisla1.append(cislo)
```

Lokální proměnná „cisla“ zastiňuje globální proměnnou „cisla“

```
moje_cisla = [1, 3, -5]
pridej_cislo_do_seznamu(moje_cisla, 8)
```

Číslo je přidáno do globální proměnné „cisla1“

# ORGANIZACE KÓDU

## DOKUMENTACE

- Docstrings jsou vysvětlující komentáře, které jsou ohraničené trojicí uvozovek

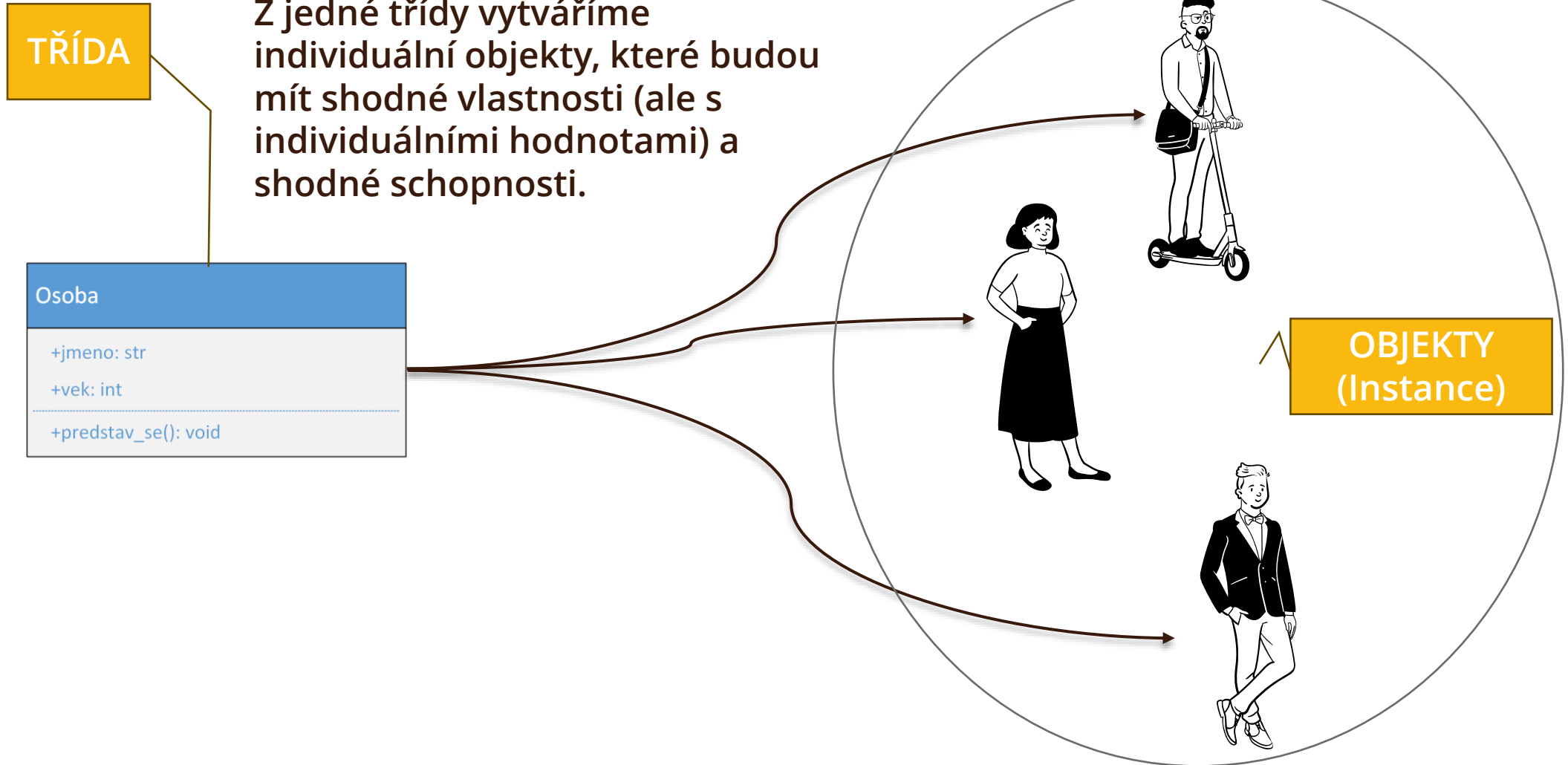
```
def secti(scitanec, scitanec1):
 """
 Tato funkce sečte dvě čísla a vrátí výsledek.

 Parametry:
 scitanec (int nebo float): První číslo k sečtení.
 scitanec1 (int nebo float): Druhé číslo k sečtení.

 Návratová hodnota:
 int nebo float: Součet scitanec a scitanec1.
 """
 return scitanec + scitanec1
```

# OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

## INSTANCE A TŘÍDY





# OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

## DATOVÉ ATRIBUTY

TŘÍDA

Osoba

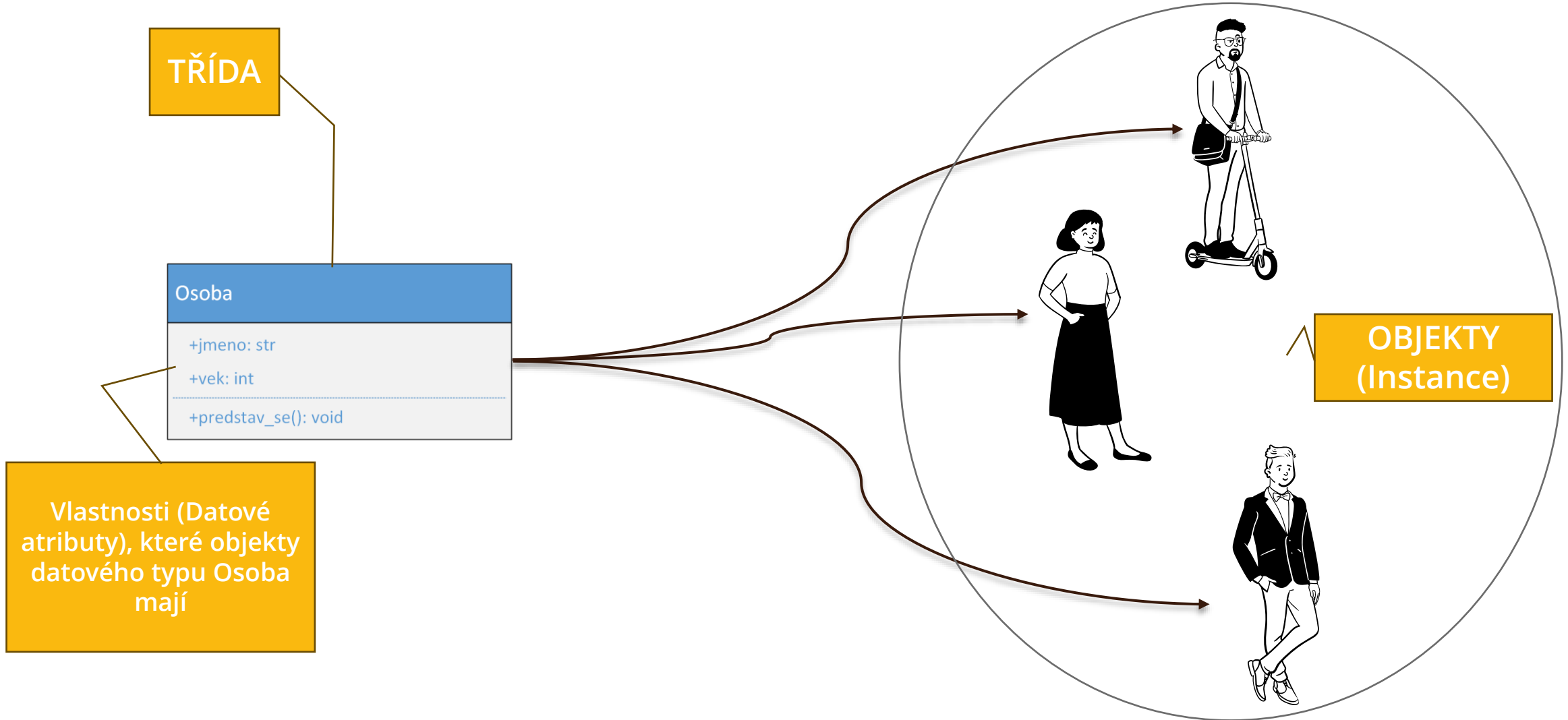
+jmeno: str

+vek: int

+predstav\_se(): void

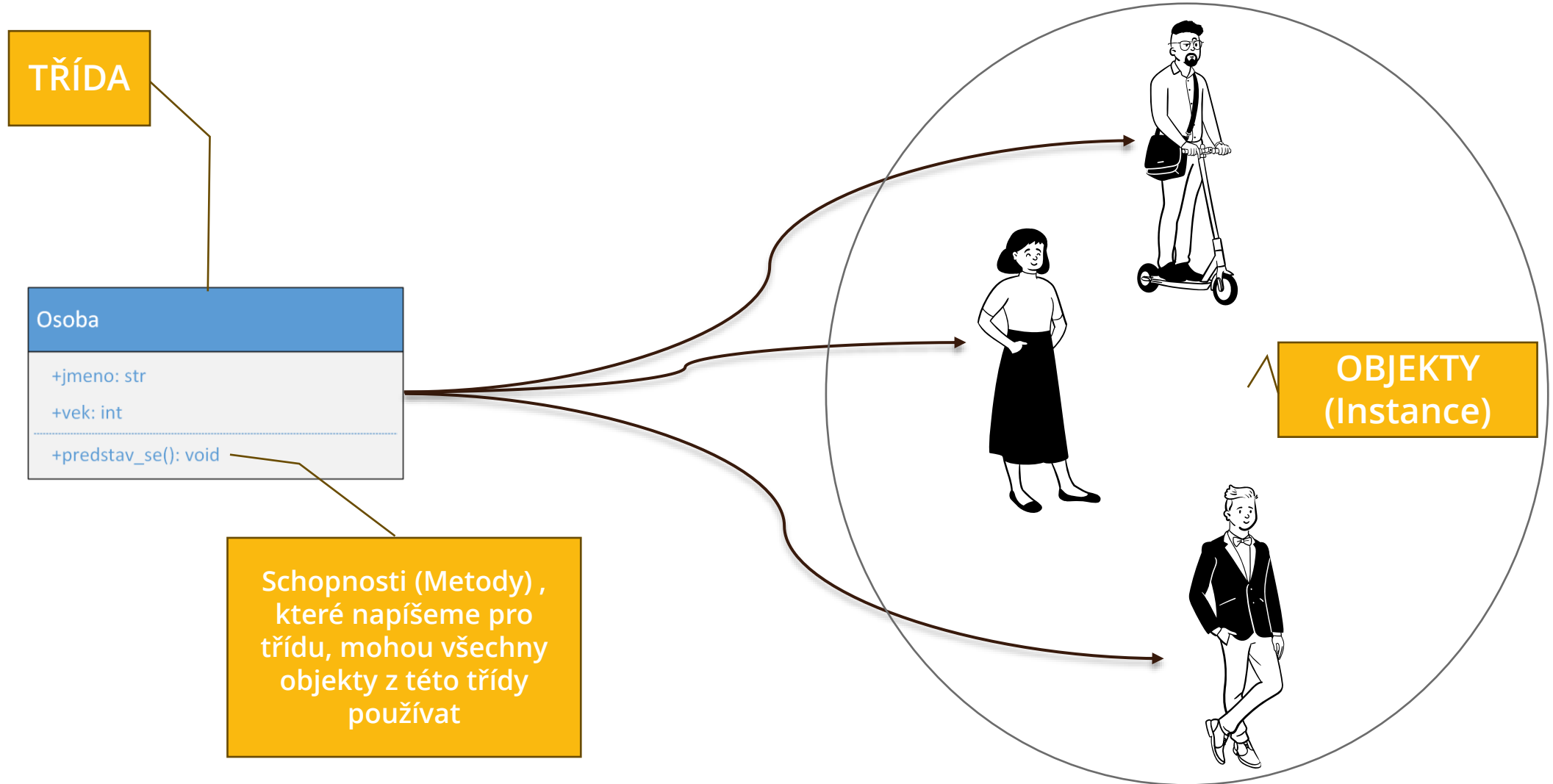
Vlastnosti (Datové atributy), které objekty datového typu Osoba mají

OBJEKTY  
(Instance)



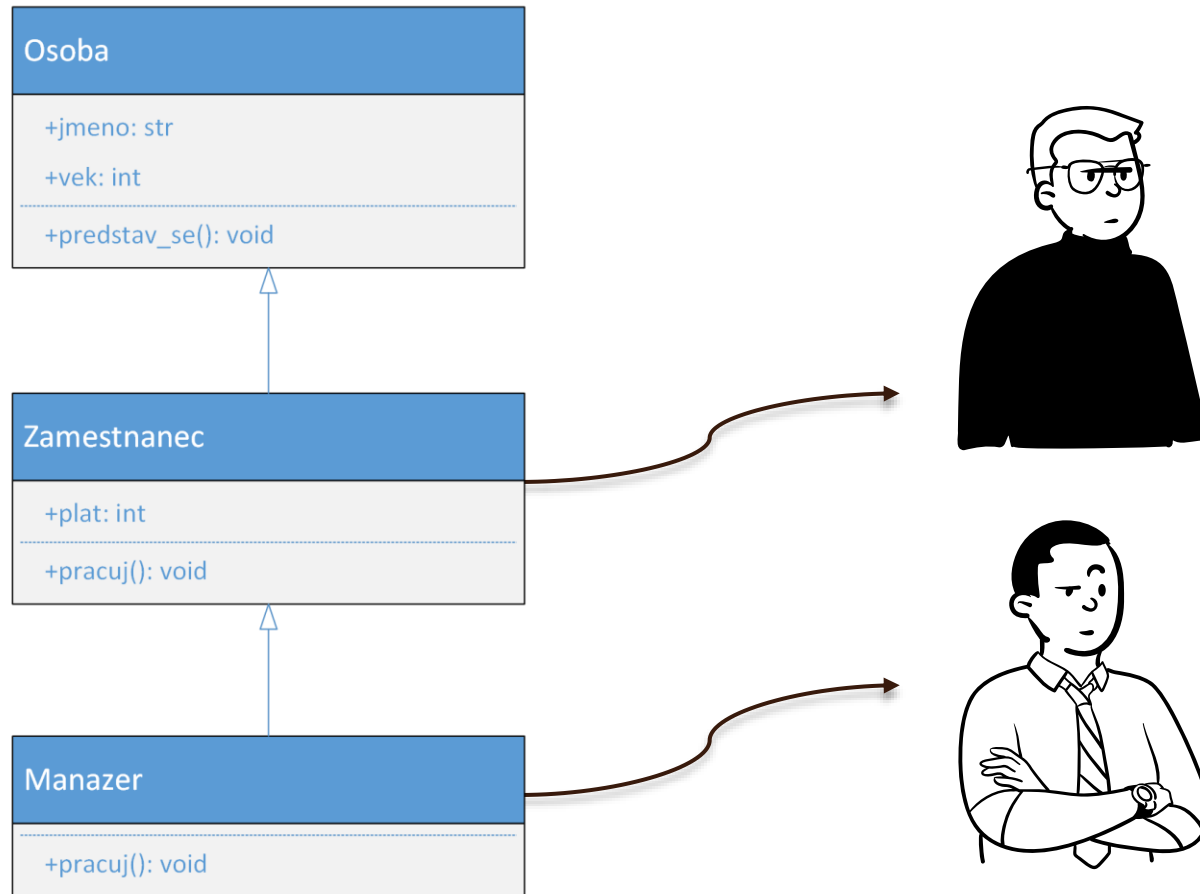
# OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

## METODY



# OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

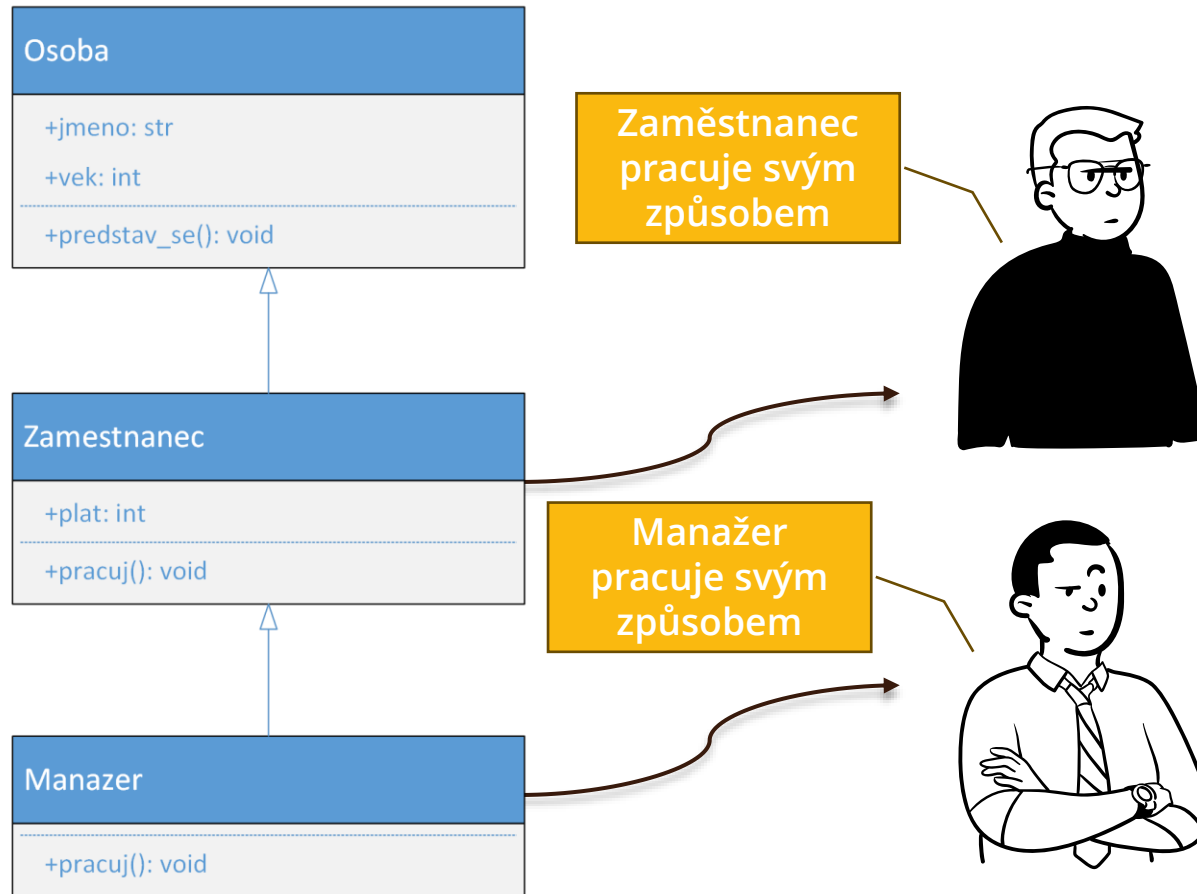
## DĚDIČNOST



Dědící třída získá všechny datové atributy a metody z předka. Může navíc přidat i další vlastní datové atributy a metody nebo už existující metody přepsat (napsat novou implementaci)

# OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

## POLYMORFISMUS



Pokud necháme skupinu složenou ze zaměstnanců pracovat, polymorfismus zajistí to, že metoda „pracuj“ se bude chovat rozdílně.


# OBJEKTOVÝ MODEL V PYTHONU

## TŘÍDA

- Třidu vytvoříme pomocí klíčového slova `class` následovaným názvem třídy
- Uvnitř třídy definujeme metody. Ty mohou pracovat s datovými atributy.
- Z této třídy bude možné vytvářet jednotlivé objekty, které budou moci používat její metody a budou mít individuální hodnoty v datových attributech

```
class Osoba:
 def __init__(self, jmeno, vek):
 self.jmeno = jmeno
 self.vek = vek

 def predstav_se(self):
 print("Jmenuji se " + self.jmeno + " a je mi " + str(self.vek) + ".")
```



- Vytvořené objektu z třídy

```
karel = Osoba("Karel", 20)
```

# OBJEKTOVÝ MODEL V PYTHONU

## METODY

- Metody mají stejnou syntaxi jako funkce, začínají klíčovým slovem def
- Prvním parametrem metod je klíčové slovo self. To odkazuje k určitému objektu, který z třídy vznikne a umožňuje metodě s objektem pracovat.

```
class Osoba:
 def __init__(self, jmeno, vek):
 self.jmeno = jmeno
 self.vek = vek

 def predstav_se(self):
 print("Jmenuji se " + self.jmeno + " a je mi " + str(self.vek) + ".")
```

# OBJEKTOVÝ MODEL V PYTHONU

## KONSTRUKTOR

- Pomocí speciální metody `__init__` nastavíme parametry, za které musíme dosadit, když z dané třídy vytváříme objekt
- Pokud klíčové slovo `self` použijeme společně s proměnnými, pracujeme tak s datovými atributy (proměnnými objektu)
- Typickým účelem speciální metody `__init__` je vytvořit datové atributy a nastavit jim počáteční hodnoty

```
class Osoba:
 def __init__(self, jmeno, vek):
 self.jmeno = jmeno
 self.vek = vek

 def predstav_se(self):
 print("Jmenuji se " + self.jmeno + " a je mi " + str(self.vek) + ".")
```

# OBJEKTOVÝ MODEL V PYTHONU

## VYBRANÉ SPECIÁLNÍ METODY

- Ve speciální metodě `__str__` implementuje vrácení textu, který bude získán, pokud budeme tisknout celý objekt
- Ve speciální metodě `__repr__` implementuje vrácení hodnoty (může se jednat ale nemusí o text), který reprezentuje celý objekt
- Ve speciální metodě `__eq__` implementujeme logiku, která porovná, zda se objekt rovná objektu jinému

```
def __str__(self):
 return "Osoba{jmeno: " + self.jmeno + ", vek: " + str(self.vek) + "}"
```

```
def __repr__(self):
 return self.jmeno
```

```
def __eq__(self, other):
 return self.vek == other.vek
```



# OBJEKTOVÝ MODEL V PYTHONU

## DĚDIČNOST

- Pokud za název třídy do kulatých závorek napíšeme název jiné třídy, zdědí tato třída všechny metody z této třídy
- V konstruktoru třídy pomocí metody `super()` můžeme zavolat konstruktor třídy, z které dědíme
- Do třídy můžeme přidat nové metody nebo přepsat existující

```
class Zamestnanec(Osoba):
```

```
 def __init__(self, jmeno, vek, plat):
 super().__init__(jmeno, vek)
 self.plat = plat
```

```
 def pracuj(self):
 print("Zamestnanec " + self.jmeno + " tvorí webovou aplikaci. ")
```

Nově vytvořená  
metoda „pracuj()“

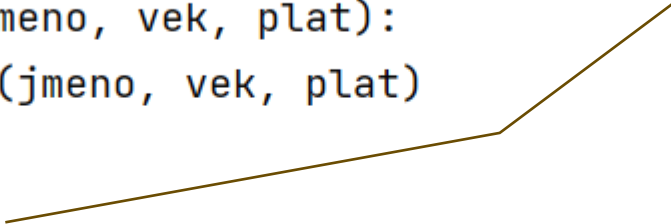
# OBJEKTOVÝ MODEL V PYTHONU

## VÍCEÚROVŇOVÁ DĚDIČNOST

- I třída, která dědí, může být použita jako předek nové další třídy

```
class Manazer(Zamestnanec):
 def __init__(self, jmeno, vek, plat):
 super().__init__(jmeno, vek, plat)

 def pracuj(self):
 print("Manazer " + self.jmeno + " jedna se zakazniky a vede porady. ")
```



Zde je metoda z třídy  
Osoba „pracuj()“  
přepsána

# OBJEKTOVÝ MODEL V PYTHONU

## POLYMORFISMUS

- Na stejné proměnné zavoláme stejnou metodou, ale protože tato proměnná postupně ukazuje na různé objekty a každý objekt může mít tuto metodu implementovanou jinak, výsledek může být rozdílný

```
jirka = Zamestnanec("Jirka", 20, 40000)
```

```
josef = Manazer("Josef", 40, 70000)
```

```
jitka = Zamestnanec("Jitka", 25, 50000)
```

```
zamestnanci = [jirka, josef, jitka]
```

```
for zamestnanec in zamestnanci:
```

```
 zamestnanec.pracuuj()
```


# OBJEKTOVÝ MODEL V PYTHONU

## DUCK-TYPING

- To, že na stejné proměnná jde volat stejná metoda různě implementovaná v různých objektech, není v Pythonu podmíněno dědičností. V Pythonu stačí, že různé objekty tuto metodu mají.

```
class Kocka():
 def vydej_zvuk(self):
 print("Mnau")
```

```
class Pes():
 def vydej_zvuk(self):
 print("Haf")
```



```
kocka = Kocka()
pes = Pes()
zvirata = [kocka, pes]
```

```
for zvire in zvirata:
 zvire.vydej_zvuk()
```

# OBJEKTOVÝ MODEL V PYTHONU

## DYNAMICKÉ VLASTNOSTI

- Do objektu je možné dynamicky přidat další vlastnosti, které nebyly součástí třídy, z které byl objekt vytvořen

```
karel = Osoba("Karel", 20)
karel.vyska = 180
print(karel.vyska)
```

# OBJEKTOVÝ MODEL V PYTHONU

## VOLÁNÍ METOD PŘEDKA

- Metody potomka mohou využít metodou `super()`, aby zavolaly metodu předka

```
class Zamestnanec(Osoba):
 def __init__(self, jmeno, vek, plat):
 super().__init__(jmeno, vek)
 self.plat = plat

 def pracuj(self):
 print("Zamestnanec " + self.jmeno + " tvorí webovou aplikaci. ")

 def predstav_se(self):
 super().predstav_se()
 print("Muj je " + str(self.plat) + ".")
```

# STANDARDNÍ KNIHOVNA

## VYUŽÍVÁNÍ HOTOVÝCH NÁSTROJŮ

- V distribuci Pythonu je standartně přítomná rozsáhlá knihovna dalších komponent (tuto knihovnu nemusíme doinstalovat)
- Další balíčky můžeme zpřístupníme pomoc klíčového slova import
- Například matematický balíčky math nebo random

```
import math
```

```
import random
```

```
cislo = 5
```

```
druha_mocnina = math.pow(5, 2)
```

```
odmocnina = math.sqrt(druha_mocnina)
```

```
nahodne_cislo = random.randint(10, 100)
```

```
cisla = [10, 8, 7, 12]
```

```
vylosovane_cislo = random.choice(cisla)
```

# STANDARDNÍ KNIHOVNA

## UKLÁDÁNÍ DAT

- Pomoci modulu pickle můžeme objekty serializovat a deserializovat

```
import pickle
```

```
karel = Osoba("Karel", 20)
with open("osoba.bin", "wb") as f:
 pickle.dump(karel, f)
```

```
karel = None
print(karel)
```

```
with open("osoba.bin", "rb") as f:
 karel = pickle.load(f)
```

```
print(karel)
```



# STANDARDNÍ KNIHOVNA

## PRÁCE S HTTP

- K odeslání požadavku (request) na server můžeme použít balíček http.client
- Ke parsování (syntaktické analýze) dat odpovědi (response) potom balíček json

```
import http.client
import json

host = "v6.exchangerate-api.com"
conn = http.client.HTTPSConnection(host)
conn.request("GET", "/v6/5d05c740a5687982d8e28e9f/latest/EUR")
response = conn.getresponse()
print(response.status, response.reason)
body = response.read()
print(str(body))
y = json.loads(body)
print(y)
print(y["time_last_update_utc"])
print(y["conversion_rates"]["CZK"])
```

# DISKUZE A DALŠÍ ZDROJE

## VLASTNÍ TÉMATA, DOKUMENTACE, ČESKÁ KOMUNITA

### Dokumentace

- Oficiální dokumentace <https://docs.python.org/3/>
- Standardní knihovna <https://docs.python.org/3/library/>

### Česká komunita

- Česká komunita <https://python.cz/>
- PyLadies <https://pyladies.cz/>

## Školení ICT Pro probíhala již ve 30 zemích světa na 4 kontinentech

Už více než 25 let pomáháme firmám a jednotlivcům růst. Po odborné i lidské stránce.

Rozvíjíme, inspirujeme a motivujeme: Pro efektivnější a kvalitnější práci.

Pro naplněný a spokojený život. A snad i pro lepší svět.



Na cestě k pozitivním změnám vás provází lektoři, co skutečně naučí a nadchnou k další práci na vašem rozvoji.

A to tak, abyste dokázali zase o něco více využít svůj potenciál.