

# Základy jazyka

Jiří Zacpal



KATEDRA INFORMATIKY  
UNIVERZITA PALACKÉHO V OLOMOUCI

KMI/ZPP1 Základy programování v Pythonu 1

# Doporučená literatura



1. **Mark Lutz: *Learning Python*.**
2. Libovolné další učebnice jazyka Python.

# Požadavky na zápočet

- Pro zápočet je potřeba:
  1. mít alespoň 75% docházku,
  2. získat **24 bodů**:
    - 2 domácí úkoly, každý za 4 body,
    - 24 bodů za úkoly na cvičeních.
- Kdo již umí programovat v Pythonu (nebo si to myslí), může příští týden zkusit napsat vstupní test.

# Konzultace



- v pracovně 5.071,
- každou středu (12.00 – 13.00),
- jindy po vzájemné domluvě,
- email: [jiri.zacpal@upol.cz](mailto:jiri.zacpal@upol.cz),
- MS Teams

Základní pojmy

# Program



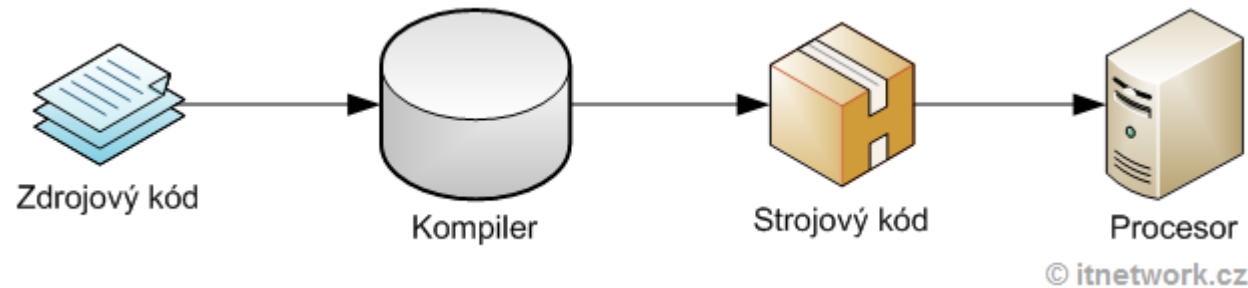
- zapsaný **postup**, jak má **procesor**, pro nějž je program určen, splnit zadanou **úlohu**,
- procesor:
  - počítač,
  - člověk,
  - ...
- program je zapsán v programovacím jazyku:
  - **strojový kód**,
  - **assembler** - zápis příkazů pomocí zkratk (jazyk symbolických instrukcí) -> překladač -> strojový kód,
  - **vyšší programovací jazyky**.

# Syntaxe – sémantika - paradigma

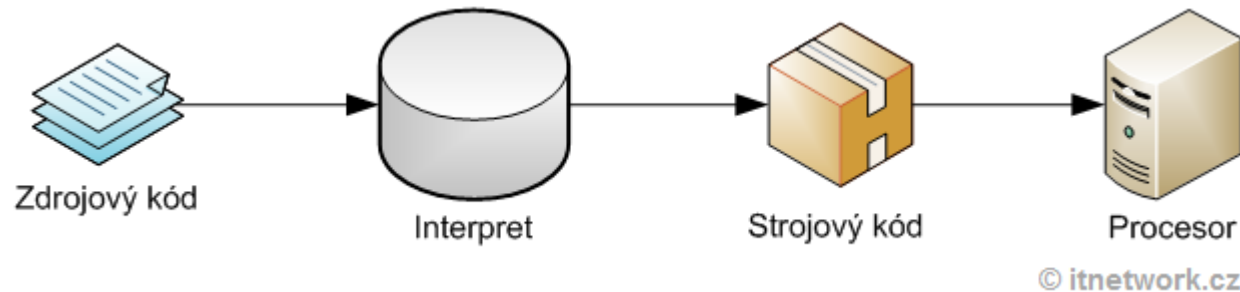
- **Syntaxe**
  - definuje, jak se zapisují jednotlivé konstrukce jazyka,
  - soubor pravidel, jaké kombinace symbolů vedou ke správně zapsanému programu.
- **Sémantika**
  - popisuje význam syntakticky správných textů v daném programovacím jazyce,
  - dále popisuje způsob jejich interpretace.
- **Paradigma**
  - označuje celkový přístup programátora k návrhu programu:
    - modulární programování,
    - strukturované programování,
    - funkcionální programování,
    - objektově orientované programování.

Programovací jazyky



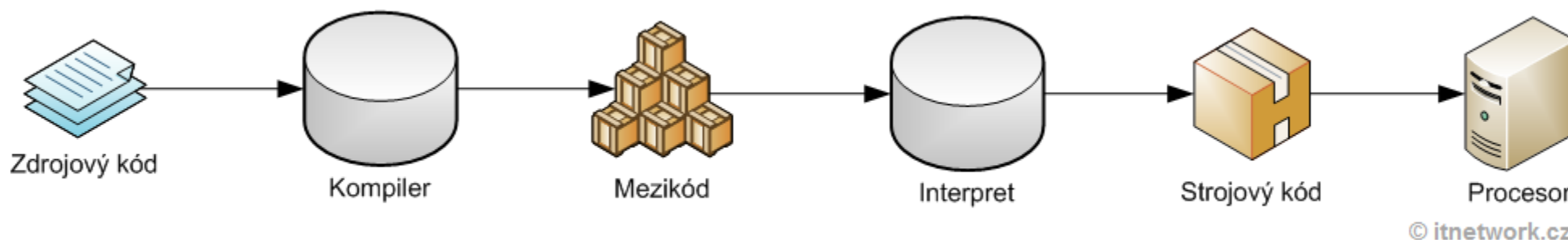


- **Výhody:**
  - **Rychlost.**
  - **Nepřístupnost zdroj. kódu** - Program se šíří již zkompilovaný, není jej možné jednoduše modifikovat pokud zároveň nevlastníte jeho zdroj. kód.
  - **Snadné odhalení chyb ve zdroj. kódu** - Pokud zdrojový kód obsahuje chybu, celý proces kompilace spadne a programátor je s chybou seznámen. To značně zjednodušuje vývoj.
- **Nevýhody:**
  - **Závislost na platformě.**
  - **Nemožnost editace.**
  - **Memory management** - Vzhledem k tomu, že počítač danému programu nerozumí a jen mechanicky vykonává instrukce, můžeme se někdy setkat s velmi nepříjemnými chybami s přetečením paměti. Kompilované jazyky obvykle nemají automatickou správu paměti a jsou to jazyky nižší (s nižším komfortem pro programátora). Běhové chyby způsobené zejména špatnou správou paměti se kompilací neodhalí.
- Příklad: [C](#), [C++](#).



- **Výhody:**
  - **Přenositelnost.**
  - **Jednodušší vývoj** - Ve vyšších jazycích jsme odstíněni od správy paměti, kterou za nás dělá tzv. garbage collector (řekneme si o něm v seriálu více). Často také nemusíme ani zadávat datové typy a máme k dispozici vysoce komfortní kolekce a další struktury.
  - **Stabilita** - Díky tomu, že interpret kódu rozumí, předejde chybám, které by zkompilovaný program jinak klidně vykonal. Běh interpretovaných programů je tedy určitě bezpečnější, dále umožňuje zajímavou vlastnost, tzv. reflexi, kdy program za běhu zkoumá sám sebe, ale o tom později.
  - **Jednoduchá editace.**
- **Nevýhody:**
  - **Rychlost.**
  - **Často obtížné hledání chyb** - Díky kompilaci za běhu se chyby v kódu objeví až v tu chvíli, kdy je kód spuštěn. To může být někdy velmi nepříjemné.
  - **Zranitelnost** - Protože se program šíří v podobě zdrojového kódu, každý do něj může zasahovat nebo krást jeho části.
- Příklad: [PHP](#).

# Jazyky s virtuálním strojem



## ■ Výhody:

- **Odhalení chyb ve zdrojovém kódu** - Díky kompilaci do CIL (Common Intermediate Language) jednoduše odhalíme chyby ve zdrojovém kódu.
- **Stabilita** - Díky tomu, že interpret kódu rozumí, zastaví nás před vykonáním nebezpečné operace a na chybu upozorní. Můžeme také provádět reflexi (i když pro CIL, ale od toho jsme většinou odstíněni).
- **Jednoduchý vývoj** - Máme k dispozici hitech datové struktury a knihovny, správu paměti za nás provádí garbage collector.
- **Slušná rychlost** - Rychlost se u virtuálního stroje pohybuje mezi interpretem a kompilerem. Virtuální stroj již výsledky své práce po použití nezahazuje, ale dokáže je cachovat, sám se tedy optimalizuje při čtenějších výpočtech a může dosahovat až rychlosti kompilery (Just In time Compiler). Start programu bývá pomalejší, protože stroj překládá společně využívané knihovny.
- **Málo zranitelný kód** - Aplikace se šíří jako zdrojový kód v CIL, není tedy úplně jednoduše lidsky čitelná.
- **Přenositelnost** - Asi je jasné, že hotový program poběží na každém železe, na kterém se nachází virtuální stroj. To ale není vše, my jsme dokonce nezávislí i na samotném jazyce. Na jednom projektu může dělat více lidí, jeden v C#, druhý ve [Visual Basic](#) a třetí v C++. Zdrojové kódy se poté vždy přeloží do CILu.
- Příklad: [Java](#), [C#](#), [Python](#).

Język Python

# Historie jazyka Python



- Programovací jazyk Python vytvořil v roce 1989 Guido van Rossum, který se na jeho vývoji podílí dodnes.
- Každý rok vycházejí průběžné aktualizace jazyka.
- Verze:
  - Python 2 – poslední verze 2.7.18 (20. duben 2020).
  - Python 3 – poslední verze 3.10.2 (14. ledna 2022).
- Jazyk je pojmenován po britské televizní show Monty Pythonův létající cirkus.
- V průběhu let se stal jazyk Python velice populární.

# Proč používat Python?

- Kvalitní software
  - čitelný
  - přehledný
- Vysoká produktivita práce
- Multiplatformnost
- Podpora knihoven
- Integrace komponent

# Vlastnosti jazyka Python

- Python je dynamický **interpretovaný jazyk**.
- Někdy bývá zařazován mezi takzvané **skriptovací jazyky**.
  - Skript = spustitelný textový soubor obsahující kód v nějakém skriptovacím jazyce.
- Python je hybridní jazyk (nebo také **multiparadigmatický**).
- K význačným vlastnostem jazyka Python patří jeho jednoduchost z hlediska učení. Bývá dokonce považován za jeden z nejvhodnějších programovacích jazyků pro začátečníky.
- Produktivnost z hlediska rychlosti psaní programů.

# Implementace jazyka Python



- **CPython**

- Standardní Python je implementován v jazyce C, tato implementace je označována CPython. V ní probíhá další vývoj jazyka Python. Verze jazyka Python jsou zveřejňovány jak v podobě zdrojového kódu, tak v podobě přeložených instalačních balíčků pro různé cílové platformy.

- **Jython**

- Jython je implementace Pythonu pro prostředí JVM. Je implementován v jazyce Java.

- **IronPython**

- IronPython je implementace Pythonu pro prostředí .NET/Mono. Za výhody lze považovat to, že se Python tímto stává jedním z jazyků pro platformu .NET. To současně znamená, že jej lze přímo využívat ve všech jazycích platformy .NET.

- **Brython**

- Brython je implementace Pythonu 3 v JavaScriptu. Jejím cílem je umožnit ve webovém prohlížeči programovat v jazyce Pythonu místo v JavaScriptu.

- **PyPy**

- alternativní interpret jazyka Python, který je zaměřen na výkon. Poslední verze PyPy implementuje Python 2.7.13 a 3.6.9.



Vývojová prostředí

# Program Python



- Instalační program na adrese <https://www.python.org/>
- Napsat program v jakémkoliv textovém editoru.
- Spustit program v shellu.

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36_64>python.exe  
d:/priklad.py  
Zadej cislo:15  
Zadej n:2  
Mocnina: 225
```

# Interpet



- interpret se spustí příkaz python v příkazovém řádku,
- potom je možné zadávat příkazy.

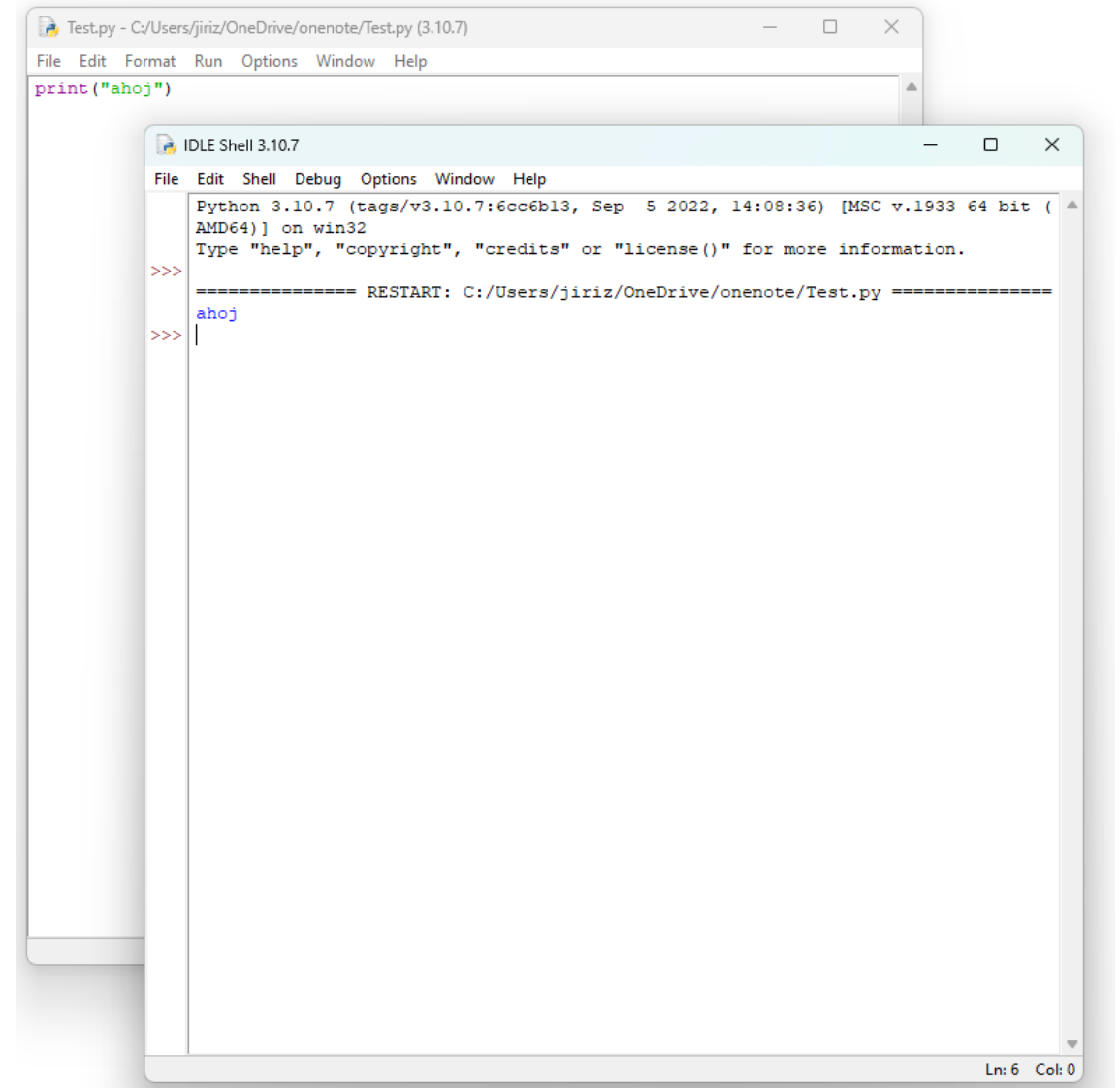
```
C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows [Version 10.0.22622.575]
(c) Microsoft Corporation. Všechna práva vyhrazena.

C:\Windows\System32>python
Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep  5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 24+56
80
>>> _
```

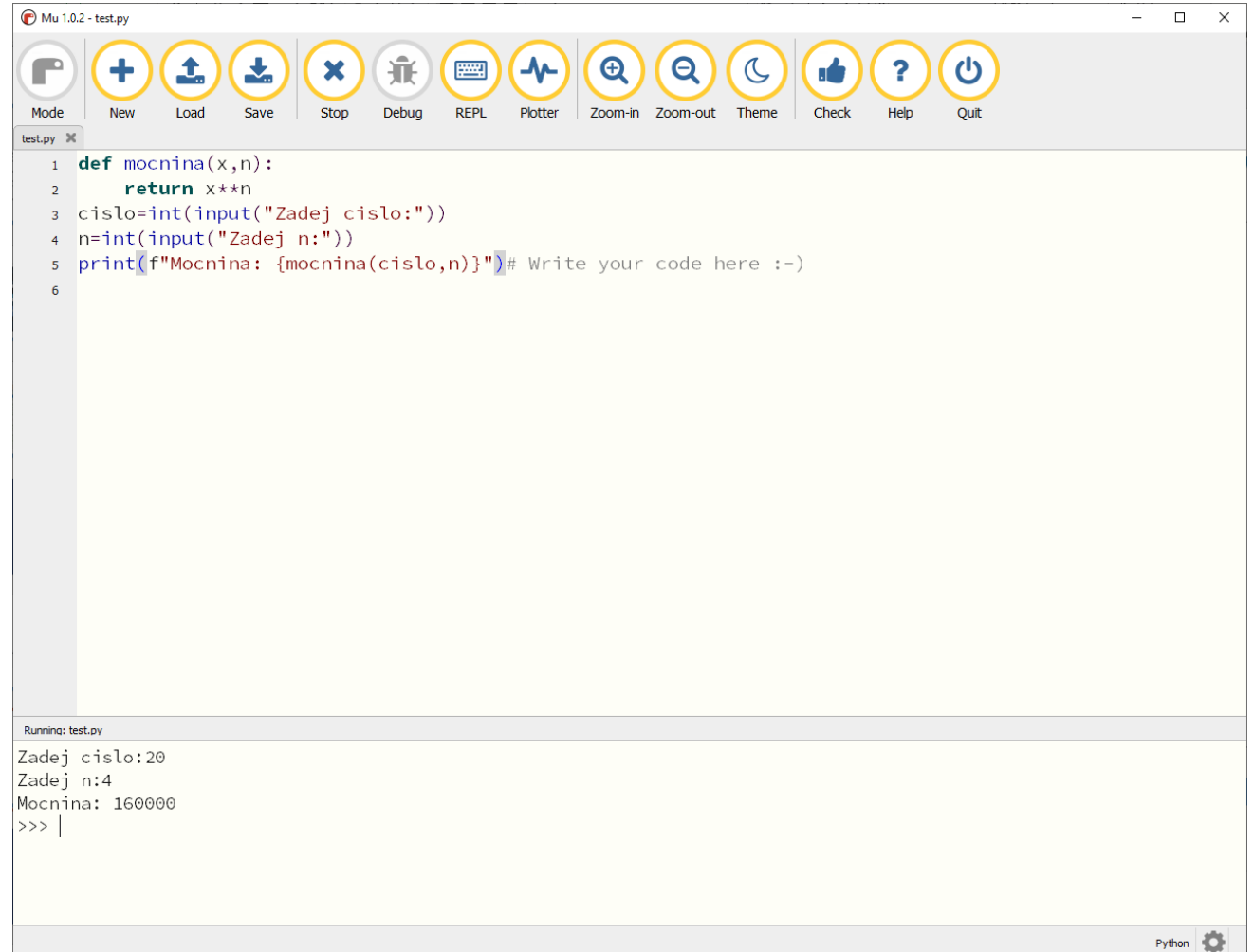
# Prostředí IDLE



- multiokenní prostředí, jehož okna mohou pracovat v jednom ze dvou režimů:
  - **interaktivní** – přímo komunikujeme s interpretem jazyka,
  - **editační** – editujeme zdrojový soubor, který pak v interaktivním použijeme.



- Editace
- Spouštění
- Ladění



The screenshot shows the Mu Python IDE interface. The title bar reads "Mu 1.0.2 - test.py". The toolbar contains icons for Mode, New, Load, Save, Stop, Debug, REPL, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The main editor area displays the following Python code:

```
1 def mocnina(x,n):  
2     return x**n  
3 cislo=int(input("Zadej cislo:"))  
4 n=int(input("Zadej n:"))  
5 print(f"Mocnina: {mocnina(cislo,n)}")# Write your code here :-)  
6
```

Below the editor, the REPL window shows the execution output:

```
Running: test.py  
Zadej cislo:20  
Zadej n:4  
Mocnina: 160000  
>>> |
```

The bottom right corner of the window shows "Python" and a settings gear icon.

# Visual Studio Code



- Lze použít na různé jazyky.
- Editace, ladění, spuštění.
- Rozšíření – extension.

The screenshot displays the Visual Studio Code editor with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The left sidebar shows icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The main editor area has two tabs: 'příklad.py' (active) and 'zpc1\_02\_typy\_operatory.c'. The 'příklad.py' tab contains the following Python code:

```
1 def mocnina(x,n):
2     return x**n
3 cislo=int(input("Zadej cislo:"))
4 n=int(input("Zadej n:"))
5 print(f"Mocnina: {mocnina(cislo,n)}")
```

Below the editor, the 'TERMINAL' panel is open, showing the command prompt output for running the script:

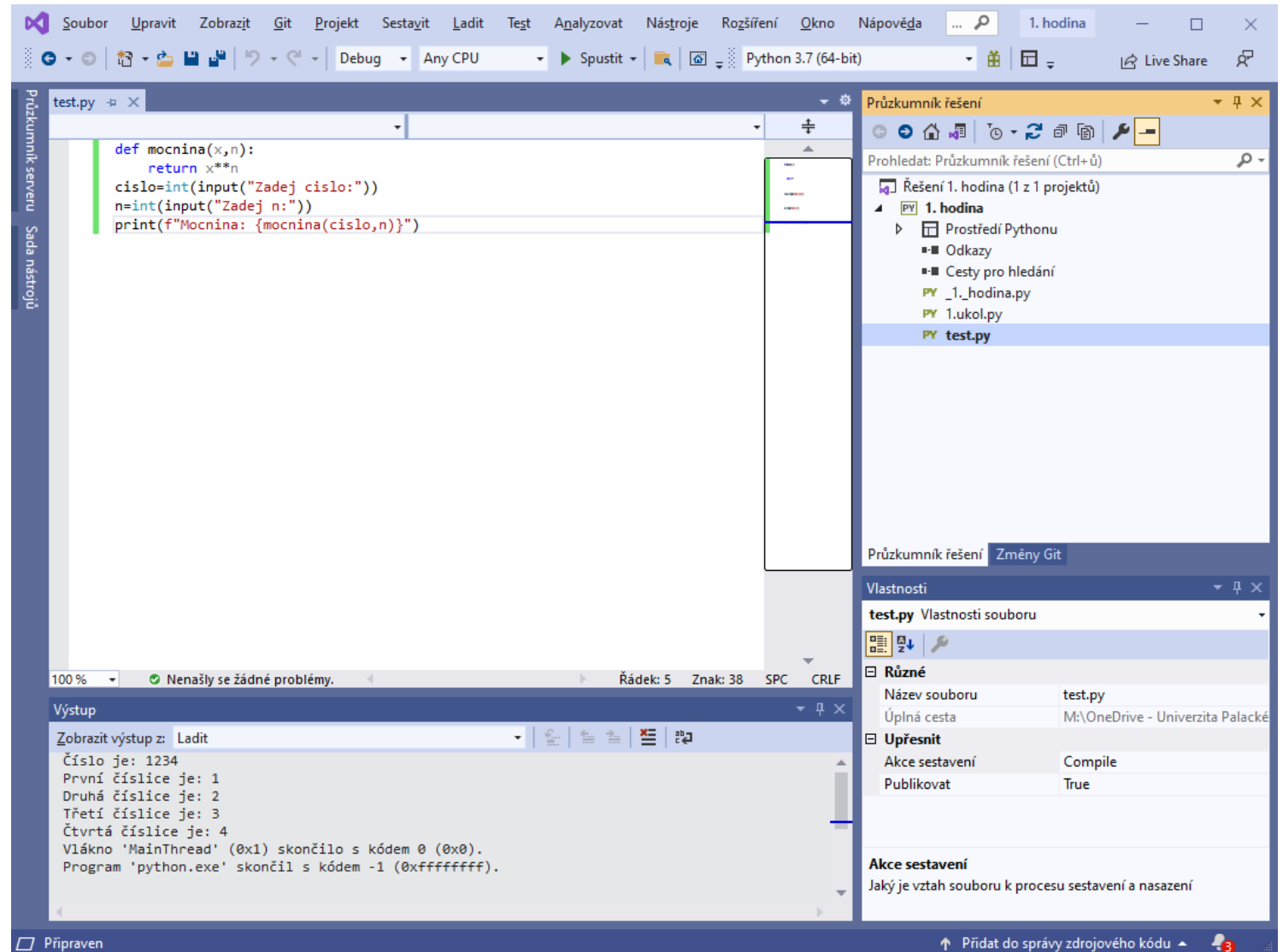
```
PS M:\OneDrive - Univerzita Palackého v Olomouci\dokumenty_pracovni\vyuka\ZS\xzpp1_zaklady_programovani_v_pythonu_1\priklady\1. hodina> .& 'C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe' 'c:\Users\Jirka\.vscode\extensions\ms-python.python-2021.6.944021595\pythonFiles\lib\python\debugpy\launcher' '58498' '--' 'm:\OneDrive - Univerzita Palackého v Olomouci\dokumenty_pracovni\vyuka\ZS\xzpp1_zaklady_programovani_v_pythonu_1\priklady\1. hodina\priklad.py'
Zadej cislo:20
Zadej n:4
Mocnina: 160000
PS M:\OneDrive - Univerzita Palackého v Olomouci\dokumenty_pracovni\vyuka\ZS\xzpp1_zaklady_programovani_v_pythonu_1\priklady\1. hodina>
```

The status bar at the bottom indicates 'Python 3.7.8 64-bit', '0 0 0' errors/warnings/hints, and 'Ln 5, Col 38'.

# Visual Studio



- Lze použít na různé jazyky.
- Velmi robustní nástroj.
- Editace, ladění, spuštění.



Hodnoty a proměnné



# Základní pojmy

- **hodnota**

- reprezentuje informace data,
- jsou reprezentována objekty.
- **Příklad:** číselná hodnota 123

```
>>> print(123)  
123
```

- **výraz**

- slouží pro vytváření hodnot.
- **Příklad:** 15 + 10

```
>>> print(15+10)  
25
```

- **příkaz**

- dává počítači instrukce, co má dělat,
- v Pythonu se na jeden řádek zapisuje jeden příkaz.

# Příklad 1



- Spustíte si interpret Pythonu v příkazovém řádku. Zadejte:

```
>>>12
```

```
12
```

```
>>>
```

- Postup interpretu:
  - R (read) – načtení vstupu
  - E (eval) – vyhodnocení vstupu
  - P (print) – vytisknutí výsledku
  - L (loop) – opakování procesu

```
>>> print 01
```

```
File "<stdin>", line 1
```

```
    print 01
```

```
        ^
```

```
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(01)?
```

```
>>>
```

# Příklad 1

- Zapište výraz 1+2:

```
>>> 1+2
```

```
3
```

```
>>>
```

- Zapište výraz 1+:

```
>>> 1+
```

```
File "<stdin>", line 1
```

```
1+
```

```
^
```

```
SyntaxError: invalid syntax
```

```
>>>
```

- Zapište výraz 1+2+3:

```
>>> 1+2+3
```

```
6
```

```
>>>
```

# Příklad 1

- Zapište výraz  $(1+2)+3$  a  $1+(2+3)$ :

```
>>> (1+2)+3
```

```
6
```

```
>>> 1+(2+3)
```

```
6
```

```
>>>
```

- Zapište výraz  $1+2+3$ ) a  $1+(2+3$ :

```
>>> 1+2+3)
```

```
File "<stdin>", line 1
```

```
1+2+3)
```

```
^
```

```
SyntaxError: invalid syntax
```

```
>>> 1+(2+3
```

```
... )
```

```
6
```

```
>>>
```

- entity, které mohou obsahovat hodnoty,
- každá proměnná má své jedinečné jméno, označované jako **identifikátor** proměnné,
- Identifikátory v jazyce Python jsou libovolná kombinace
  - malých písmen (a–z),
  - velkých písmen (A–Z),
  - číslic (0–9)
  - a znaku \_ (podtržítko).
- Identifikátor nesmí začínat číslicí a jako identifikátor nelze použít klíčová slova jazyka, která mají speciální význam.

# Použití proměnné



- **Typ proměnné** – co lze do proměnné vložit.
  - staticky typované a dynamicky typované jazyky.
  - Python je dynamicky typovaný jazyk. To znamená, že typ proměnné je určen typem objektu, který reprezentuje hodnotu, jež proměnná obsahuje.
  - Příklady:
    - 42 je objekt typu „celé číslo“,
    - 42.0 je objekt typu „desetinné číslo“.
- Proměnnou není potřeba deklarovat.
- Vytvoří se při prvním použití.

# Správa paměti

- zásobník
  - do proměnné se uloží přímo hodnota
  - tato je uložena na zásobníku
- halda
  - data se uloží na haldu (zvláštní oblast paměti)
  - do proměnné se uloží jen odkaz, kde data hledat
- Python používá pro ukládání dat vždy **haldu**.

Přiřazovací příkaz



# Přiřazovací příkaz



- zápis:

proměnná = výraz

- proměnná začíná existovat až po své **inicializaci** = přiřazení hodnoty,
- Python umožňuje zadat v jednom příkazu i skupinu hodnot:
  - pokud chceme jednu hodnotu přiřadit více proměnným, můžeme použít příkaz:

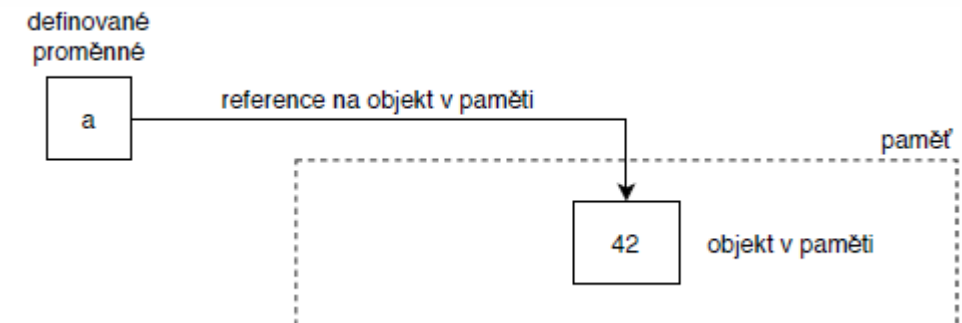
p1 = p2 = p3 = hodnota

- pokud má mít každá proměnná jinou hodnotu, použijeme příkaz:

p1, p2, p3 = h1, h2, h3

- Příklad

a=42



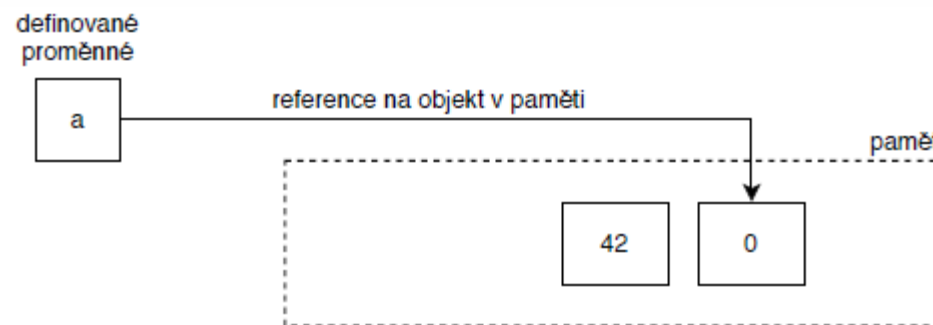
# Přiřazení a proměnná



- Proměnná, která doposud nebyla použita, se označuje jako **nedefinovaná proměnná**.
- Pokud je proměnná použita jako levý operand operátoru přiřazení poprvé, dochází k její **definici**.
- V jazyce Python dochází vždy při definici proměnné i k její **deklaraci**. Deklarace označuje přiřazení reference k proměnné, tedy **přiřazení hodnoty**.
- V jazyce Python nelze vytvořit proměnou bez hodnoty.
- První nastavení hodnoty proměnné se také označuje jako **inicializace proměnné**.
- Pokud je proměnná použita jako levý operand operátoru přiřazení již deklarována, dochází ke změně její reference.
- Příklad:

`a=42`

`a=0`



# Vlastnosti přiřazení

- Přiřazení není operátor (jako např. v jazyce C), ale příkaz.
- Tyto příkazy způsobí chybu:

```
>>> a=20+b=10
```

```
File "<stdin>", line 1
```

```
SyntaxError: can't assign to operator
```

```
>>> (a=10)+(b=2)
```

```
File "<stdin>", line 1
```

```
(a=10)+(b=2)
```

```
^
```

```
SyntaxError: invalid syntax
```

- Lze ale použít příkaz:

```
>>> a=b=10
```

- Důvodem je, že tento typ zápisu představuje **syntaktický cukr**, tedy alternativní, pohodlnější zápis jiného výrazu.

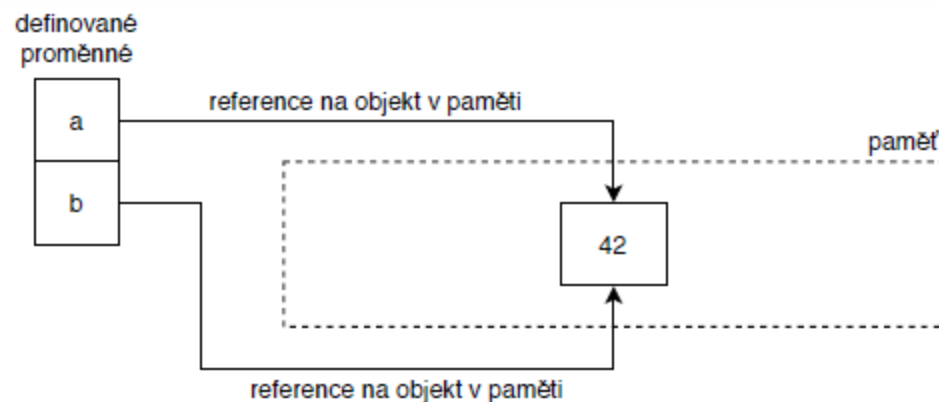
# Změna reference



- Po vykonání příkazů:

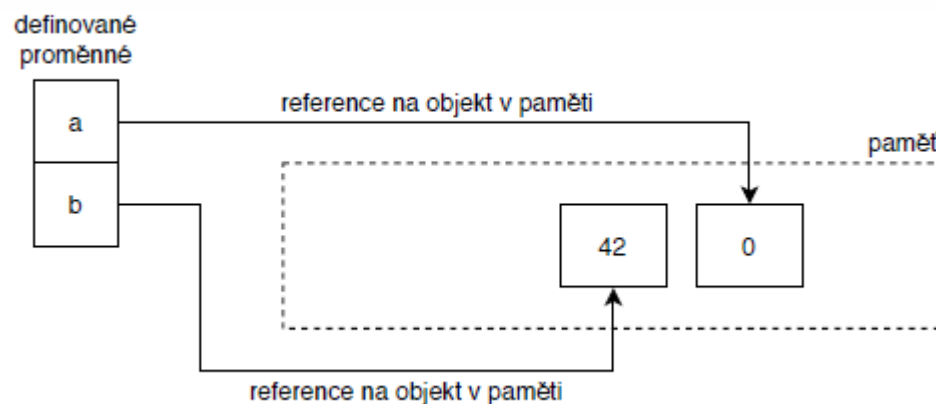
`a=42`

`b=a`



- Změna nastane po vykonání příkazu:

`a=0`



# Kombinace s aritmetickými operátory

- Přiřazení lze kombinovat s aritmetickými operátory:  
`a+=10` stejný význam jako `a=a+10`
- Kombinace
  - `+=`
  - `-=`
  - `*=`
  - `/=`
  - `%=`
  - `//=`
  - `**=`

# Příklad



- Napište příkaz:

```
>>> b=a
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'a' is not defined

```
>>>
```

Napište příkaz:

```
>>> a=123
```

```
>>> b=a
```

```
>>> b
```

```
123
```

```
>>>
```

# Příklad



- Napište příkaz:

```
>>> a,b,c=10,20,30
```

```
>>> b,c,a=a,b,c
```

```
>>> a,b,c
```

```
(30, 10, 20)
```

```
>>>
```

# Datové typy



- Existují dva základní typové systémy:
  - **Dynamický typový systém**
    - nás plně odstiňuje od toho, že proměnná má vůbec nějaký datový typ,
    - ona ho samozřejmě vnitřně má, ale jazyk to nedává najevo,
    - dynamické typování jde mnohdy tak daleko, že proměnné nemusíme ani deklarovat.,
    - jakmile do nějaké proměnné něco uložíme a jazyk zjistí, že nebyla nikdy deklarována, sám ji založí,
  - **Statický typový systém**
    - striktně vyžaduje definovat typ proměnné a tento typ je dále neměnný,
    - jakmile proměnnou jednou deklarujeme, není možné její datový typ změnit.
- Python je **dynamicky typovaný jazyk**.
- V Pythonu máme tři základní datové typy:
  - Celá čísla: int
  - Desetinná čísla: float
  - Textový řetězec: str

- celé číslo můžeme zadat jako libovolnou posloupnost desítkových číslic nezačínající nulou (s výjimkou 0),

```
>>> 123
```

```
123
```

- celá čísla mohou být libovolně velká,
- zápis velkých čísel můžeme zpřehlednit znakem \_

```
>>> 123_456_789
```

```
123456789
```

- je-li číslo záporné, vložíme před něj znak -,

```
>>> -123
```

```
-123
```

# Reálná čísla



- reálné číslo můžeme zadat ve tvaru, kdy celou a desetinnou část oddělíme .

```
>>> 1.23
```

```
1.23
```

- jiný možný způsob je zadat číslo v exponenciálním tvaru

```
>>> 1.23e3
```

```
1230.0
```

kdy tento zápis představuje číslo  $1.23 \cdot 10^3$

# Textové řetězce (stringy)

- máme dva možné typy zápisu:
  - zadávaný text uzavřeme mezi apostrofy

```
>>> 'Python'
'Python'
```
  - text uzavřeme mezi uvozovky

```
>>> "Python"
'Python'
```
  - toto lze využít, pokud text mají tvořit uvozovky nebo apostrofy:

```
>>> 'Řekni: "Ahoj světe!"'
'Řekni: "Ahoj světe!"'
```

Standardní výstup

# Visual Studio Code



- Spustíte si program Visual Studio Code.
- Vytvořte si adresář programy.
- Ve VSC zadejte příkaz File – Open folder a vyberte tento adresář.
- Vytvořte si soubor seminar01.py.

The screenshot shows the Visual Studio Code interface with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The left sidebar contains icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The main editor area displays a file named 'priklad.py' with the following Python code:

```
1 def mocnina(x,n):
2     return x**n
3 cislo=int(input("Zadej cislo:"))
4 n=int(input("Zadej n:"))
5 print(f"Mocnina: {mocnina(cislo,n)}")
```

The 'Run and Debug' sidebar on the left shows a 'Run and Debug' button and instructions: 'To customize Run and Debug, open a folder and create a launch.json file.' and 'Show all automatic debug configurations.' The bottom panel shows the 'TERMINAL' tab with the following output:

```
PS M:\OneDrive - Univerzita Palackého v Olomouci\dokumenty_pracovni\vyuka\ZS\xzpp1_zaklady_programovani_v_pythonu_1\priklady\1. hodina> . & 'C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe' 'c:\Users\Jirka\.vscode\extensions\ms-python.python-2021.6.944021595\pythonFiles\lib\python\debugpy\launcher' '58498' '--' 'm:\OneDrive - Univerzita Palackého v Olomouci\dokumenty_pracovni\vyuka\ZS\xzpp1_zaklady_programovani_v_pythonu_1\priklady\1. hodina\priklad.py'
Zadej cislo:20
Zadej n:4
Mocnina: 160000
PS M:\OneDrive - Univerzita Palackého v Olomouci\dokumenty_pracovni\vyuka\ZS\xzpp1_zaklady_programovani_v_pythonu_1\priklady\1. hodina>
```

The status bar at the bottom indicates 'Python 3.7.8 64-bit', '0 0 0', and 'Ln 5, Col 38 Spaces: 4 UTF-8 CRLF Python'.

# Výstup na obrazovce

- K výstupu na obrazovku slouží funkce

`print(entita)`

- Jednoduchý výstup:  
`print("Ahoj světe")` # zobrazí: Ahoj světe
- Výstup proměnné:  
`a = 42`  
`print("Ahoj světe", a)` # zobrazí: Ahoj svete 42
- Formátovaný výstup:  
`a = 42`  
`print(f"Ahoj svete {a}")` # zobrazí: Ahoj svete 42  
  
`pi = 3.14159265359`  
`print(f"{pi}")`  
`print(f"{pi:.2f}")` # vypíše 3.14  
`print(f"{pi:.0f}")` # vypíše 3

# Příklad



- Napište do souboru tyto příkazy:

```
a=1
```

```
print(a)
```

- Program spusťte.
- Příkazy upravte takto:

```
a=1
```

```
print(a/0)
```

- Program spusťte.
- Program skončil chybou:

```
program-1.py seminar01.py Python - C
```

```
seminar01.py > ...  
1 a=1  
2 print(a/0)
```

**Exception has occurred: ZeroDivisionError**  
division by zero

File "M:\OneDrive - Univerzita Palackého v Olomouci\dokumenty\_pracovni\vyuka\ZS\zpp1\_zaklady\_programovani\_v\_pythonu\programy\seminar01.py",  
line 2, in <module>  
 print(a/0)



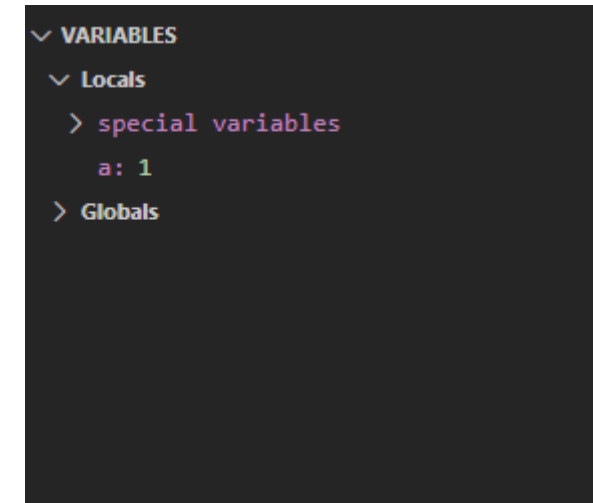
# Příklad



- Program můžeme ladit tak, že před určitý příkaz vložíme zarážku.
- Po spuštění programu, se vykonávání programu zastaví na tomto příkazu.
- K ladění můžeme využít příkazy z tohoto panelu:



- Aktuální vazby proměnných jsou v sekci Variables:
- Do kódu můžeme zapsat komentář:  
#První hodina.  
X=5 #Promenna x



# Bodovaný úkol



- Napište program, který do čtyř proměnných uloží jednotlivé číslice čtyřciferného čísla.
- Toto číslo vytiskněte (tak aby na výstupu bylo čtyřmístné číslo bez mezer).
- Potom prohodte obsah proměnných tak, aby v proměnné, kde byla 1. číslice, byla 4. číslice, ve 2. 3., atd.
- Číslo opět vytiskněte.
- Program by měl obsahovat jen 4 příkazy

## Příklad výstupu:

1234

4321