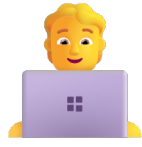
PROGRAMOVACÍ JAZYK

PYTHON

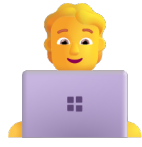
JEDNODUCHÝ. ČITELNÝ. PRAKTICKÝ.



CO JE PYTHON?

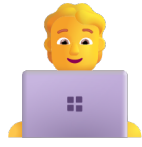
CO JE PYTHON?

- Vysoce úrovňový programovací jazyk



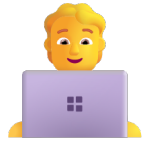
CO JE PYTHON?

- Vysoce úrovňový programovací jazyk
- Vytvořený s důrazem na čitelnost



CO JE PYTHON?

- Vysoce úrovňový programovací jazyk
- Vytvořený s důrazem na čitelnost
- Interpretovaný (nekompiluje se)

 CO JE PYTHON?

- Vysoce úrovňový programovací jazyk
- Vytvořený s důrazem na čitelnost
- Interpretovaný (nekompiluje se)
- Open-source a zdarma

 # JAK JE STARÝ PYTHON? 

 # JAK JE STARÝ PYTHON? 

<15 let

👶 JAK JE STARÝ PYTHON? 👴

<15 let

35 let

 JAK JE STARÝ PYTHON? 

<15 let

35 let

55 let

 JAK JE STARÝ PYTHON? 

<15 let

35 let

55 let

>75 let

 JAK JE STARÝ PYTHON? 

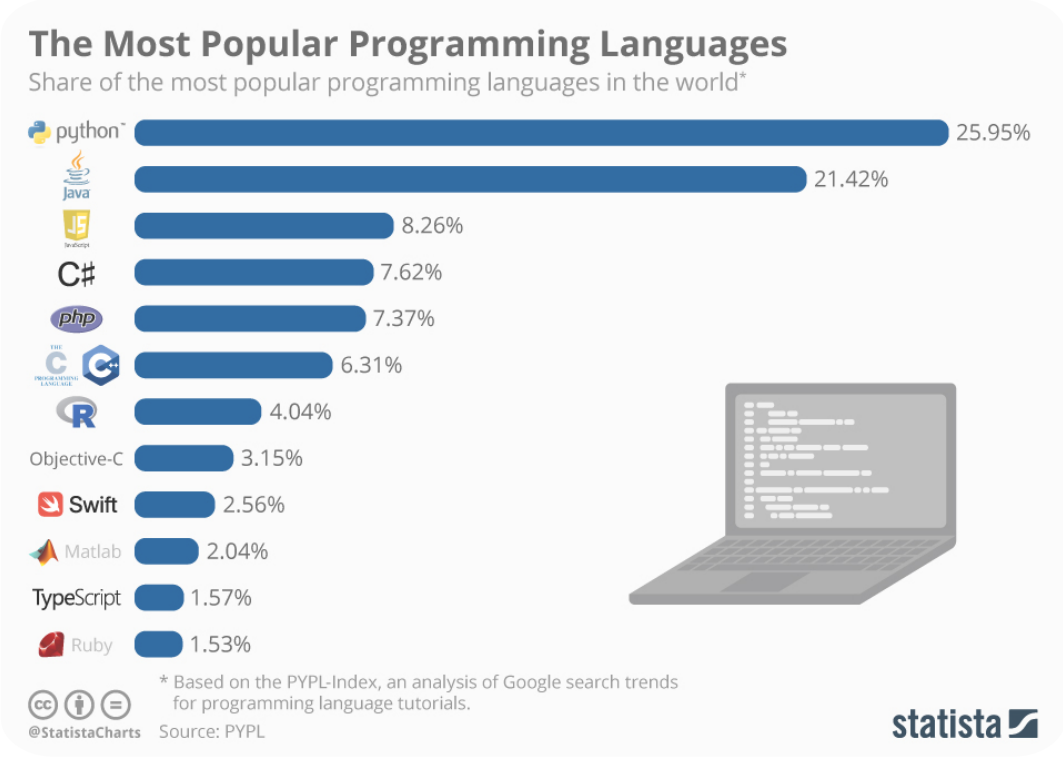
<15 let

35 let

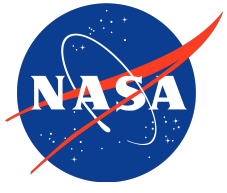
55 let

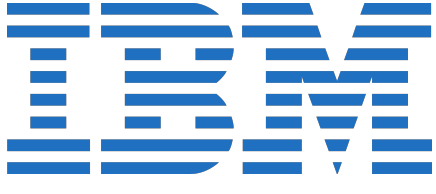
>75 let

TRENDY

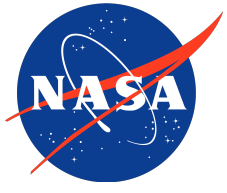

The Most Popular Programming Languages
Share of the most popular programming languages in the world*
python
25.95%
Java
21.42%
8.26%
C#
7.62%
php
7.37%
6.31%
4.04%
Objective-C
3.15%
Swift
2.56%
Matlab
2.04%
TypeScript
1.57%
Ruby
1.53%
* Based on the PYPL-Index, an analysis of Google search trends for programming language tutorials.
@StatistaCharts
Source: PYPL
statista

🤑 PYTHON SE POUŽÍVÁ TAM, KDE JE CHYBA DRAHÁ

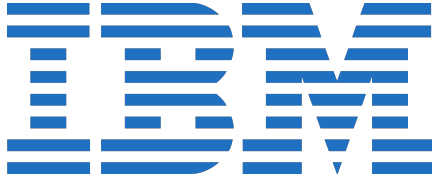




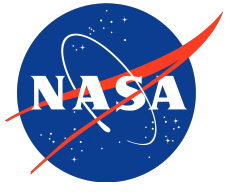
🤑 PYTHON SE POUŽÍVÁ TAM, KDE JE CHYBA DRAHÁ

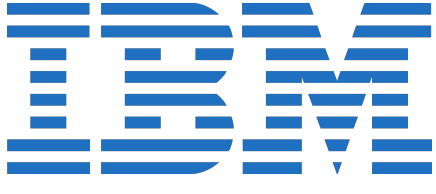


Používají Python kvůli:



🤑 PYTHON SE POUŽÍVÁ TAM, KDE JE CHYBA DRAHÁ

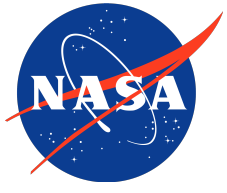


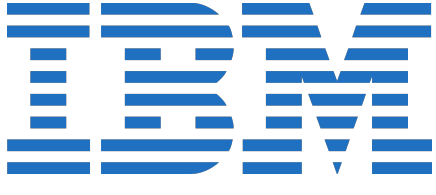


Používají Python kvůli:

- čitelnosti

🤑 PYTHON SE POUŽÍVÁ TAM, KDE JE CHYBA DRAHÁ

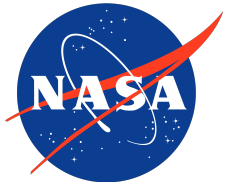


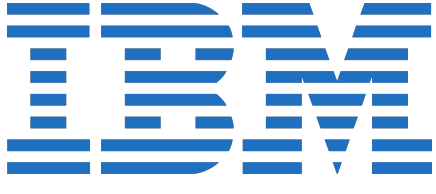


Používají Python kvůli:

- čitelnosti
- menšímu počtu bugů

🤑 PYTHON SE POUŽÍVÁ TAM, KDE JE CHYBA DRAHÁ

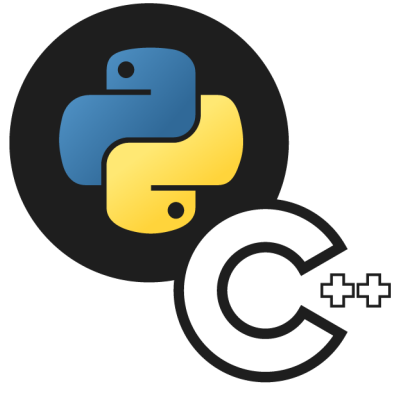




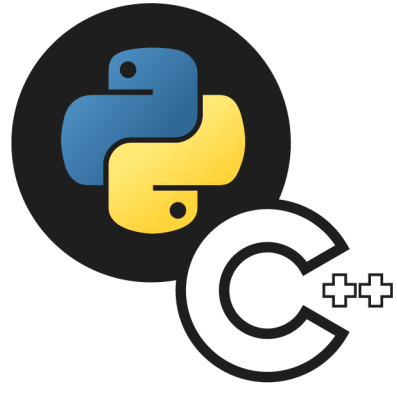
Používají Python kvůli:

- čitelnosti
- menšímu počtu bugů
- rychlému prototypování

PYTHON JE „LEPIDLO SVĚTA“

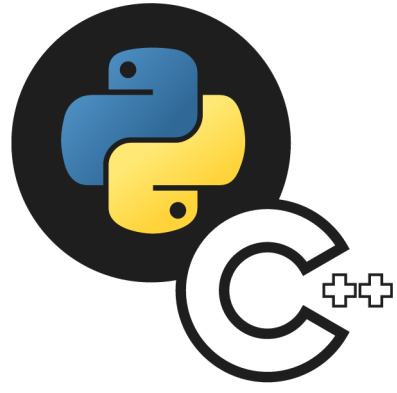


🌍 PYTHON JE „LEPIDLO SVĚTA“



🧬 Python nevznikl jako rychlý jazyk, ale jako jazyk, který spojuje ostatní věci.

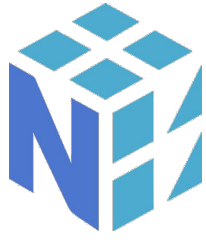
🌍 PYTHON JE „LEPIDLO SVĚTA“

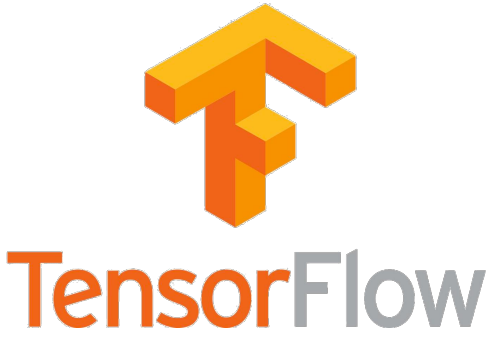


🪢 Python nevznikl jako rychlý jazyk, ale jako jazyk, který spojuje ostatní věci.

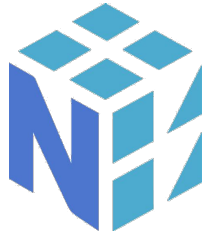
📞 V praxi Python často volá C/C++ kód, aniž si to uživatel uvědomuje.

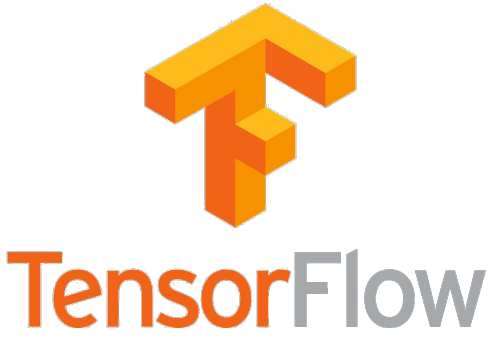
📦 VĚTŠINA „RYCHLÉHO PYTHONU“ NENÍ PYTHON





VĚTŠINA „RYCHLÉHO PYTHONU“ NENÍ PYTHON





Všechny jsou napsané v C/C++

PYTHON NEMÁ KOMPILÁTOR (TAK ÚPLNĚ)

- se nekompiluje do strojového kódu
- ale do bytecode
- který běží na Python Virtual Machine (PVM)

Bytecode	Machine Code
Bytecode is an intermediate code designed to run on a virtual machine instead of a central processing unit (CPU).	Machine code is a computer program made up of the native instructions associated with that particular computer.
The function of a bytecode is to be a format that can be executed efficiently by the virtual machine's interpreter.	Machine code is the language which all programs must be converted into before they can be run.
It is platform-independent because it can be executed on any platform using the virtual machine.	It is not platform independent meaning it cannot be run on just any platform with the same operating system.

PYTHON NEMÁ KOMPILÁTOR (TAK ÚPLNĚ)

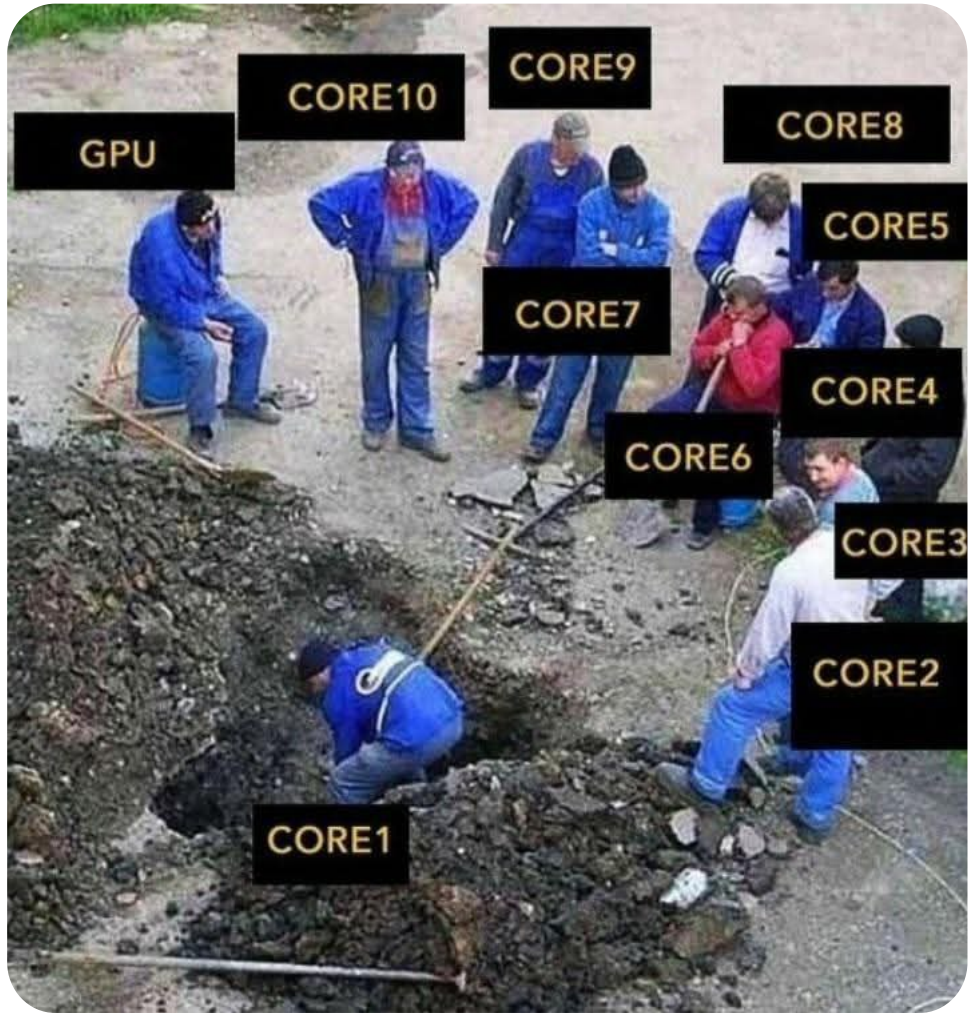
- se nekompiluje do strojového kódu
- ale do bytecode
- který běží na Python Virtual Machine (PVM)

Bytecode	Machine Code
Bytecode is an intermediate code designed to run on a virtual machine instead of a central processing unit (CPU).	Machine code is a computer program made up of the native instructions associated with that particular computer.
The function of a bytecode is to be a format that can be executed efficiently by the virtual machine's interpreter.	Machine code is the language which all programs must be converted into before they can be run.
It is platform-independent because it can be executed on any platform using the virtual machine.	It is not platform independent meaning it cannot be run on just any platform with the same operating system.

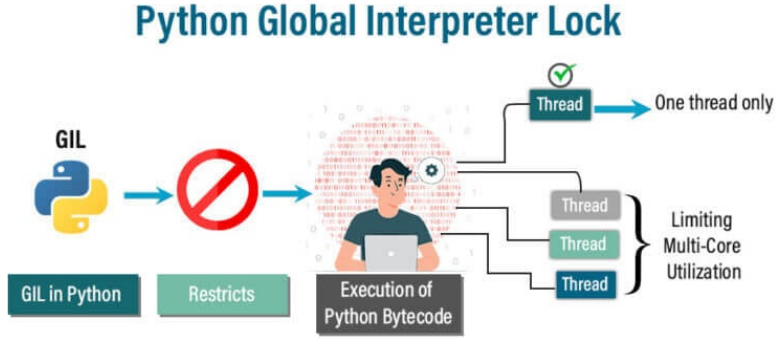
žádný **main()**, linker a hlavičky

🔐 GLOBAL INTERPRETER LOCK (GIL) NEJKONTROVERZNĚJŠÍ VĚC V PYTHONU

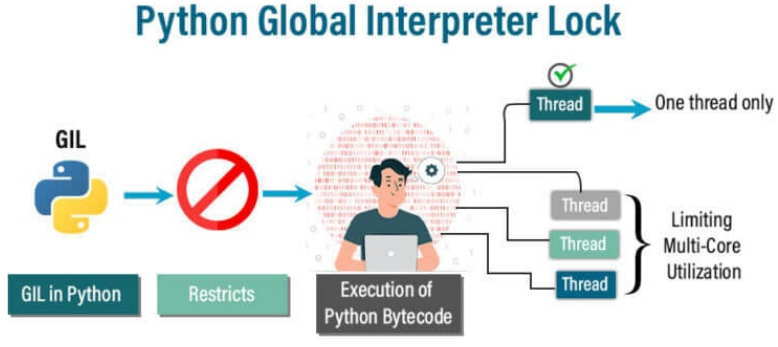
🔐 GLOBAL INTERPRETER LOCK (GIL) NEJKONTROVERZNĚJŠÍ VĚC V PYTHONU



🔐 GLOBAL INTERPRETER LOCK (GIL) NEJKONTROVERZNĚJŠÍ VĚC V PYTHONU

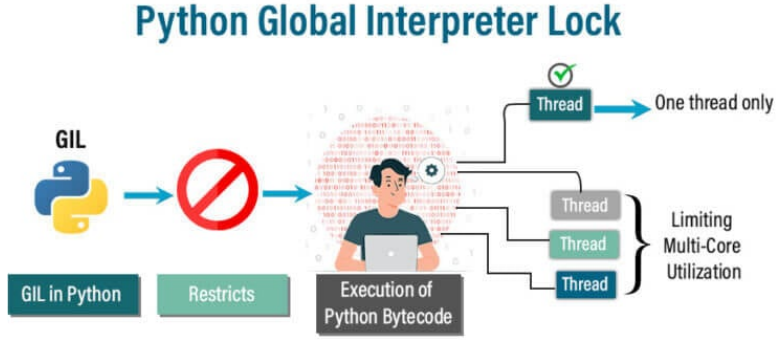


🔐 GLOBAL INTERPRETER LOCK (GIL) NEJKONTROVERZNĚJŠÍ VĚC V PYTHONU



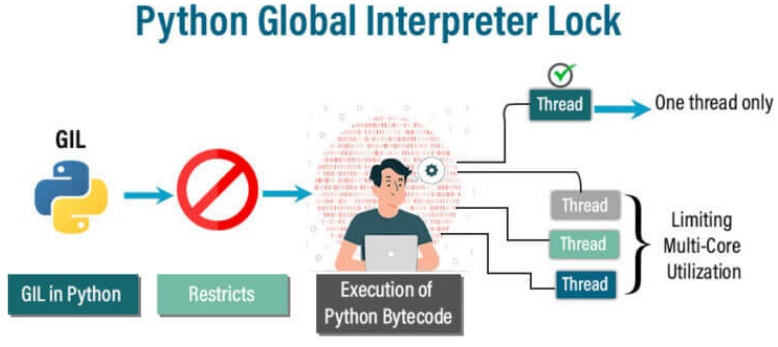
- dovoluje běžet jen jednomu vláknu Python kódu

🔐 GLOBAL INTERPRETER LOCK (GIL) NEJKONTROVERZNĚJŠÍ VĚC V PYTHONU



- dovoluje běžet jen jednomu vláknu Python kódu
- i na vícejádrovém CPU...

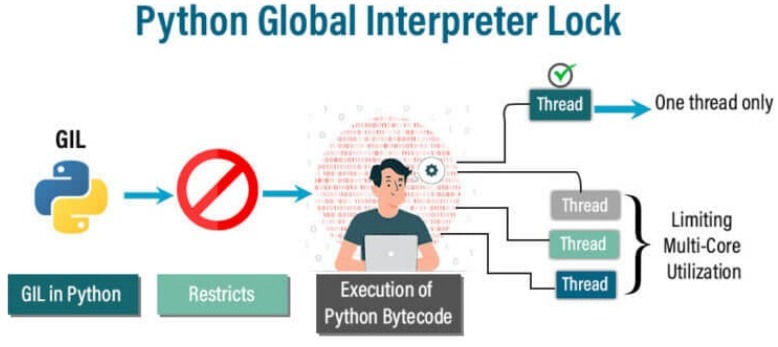
🔐 GLOBAL INTERPRETER LOCK (GIL) NEJKONTROVERZNĚJŠÍ VĚC V PYTHONU



- dovoluje běžet jen jednomu vláknu Python kódu
- i na vícejádrovém CPU…

“threading ≠ paralelismus”

🔐 GLOBAL INTERPRETER LOCK (GIL) NEJKONTROVERZNĚJŠÍ VĚC V PYTHONU



- dovoluje běžet jen jednomu vláknu Python kódu
- i na vícejádrovém CPU...

“threading ≠ paralelismus”

❌ threading ✅ multiprocessing

🏠 PYTHON BYL NAVRŽEN TAK, ABY SE ČETL NAHLAS

🏠 PYTHON BYL NAVRŽEN TAK, ABY SE ČETL NAHLAS

Python je jediný mainstream jazyk, kde:

🏠 PYTHON BYL NAVRŽEN TAK, ABY SE ČETL NAHLAS

Python je jediný mainstream jazyk, kde:

```
1 for x in data: # data = [52.3, 28.4, 0, 1.5, -142.4, 284.2]
2   if x > 0:
3     print(x)
```

🏠 PYTHON BYL NAVRŽEN TAK, ABY SE ČETL NAHLAS

Python je jediný mainstream jazyk, kde:

```
1 for x in data: # data = [52.3, 28.4, 0, 1.5, -142.4, 284.2]
2   if x > 0:
3     print(x)
```

… opravdu odpovídá tomu, jak bys to řekl/a lidsky.

🏠 PYTHON BYL NAVRŽEN TAK, ABY SE ČETL NAHLAS

Python je jediný mainstream jazyk, kde:

```
1 for x in data: # data = [52.3, 28.4, 0, 1.5, -142.4, 284.2]
2   if x > 0:
3     print(x)
```

… opravdu odpovídá tomu, jak bys to řekl/a lidsky.

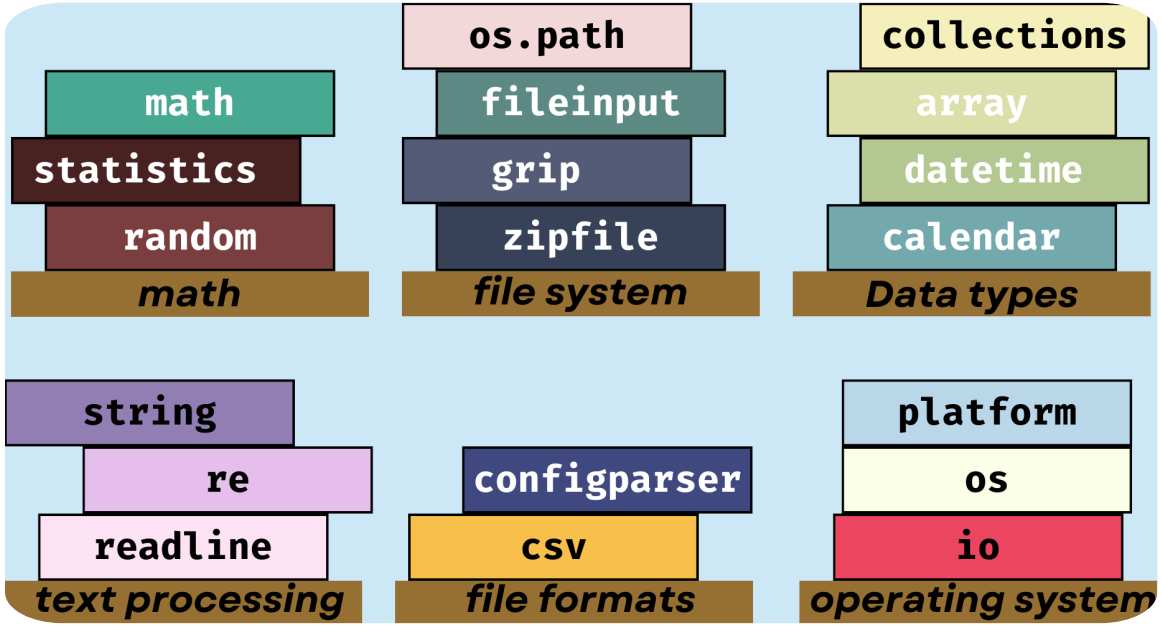
žádné {}

žádné ;

odsazení má význam

PYTHON MÁ „BATERIE V BALENÍ“

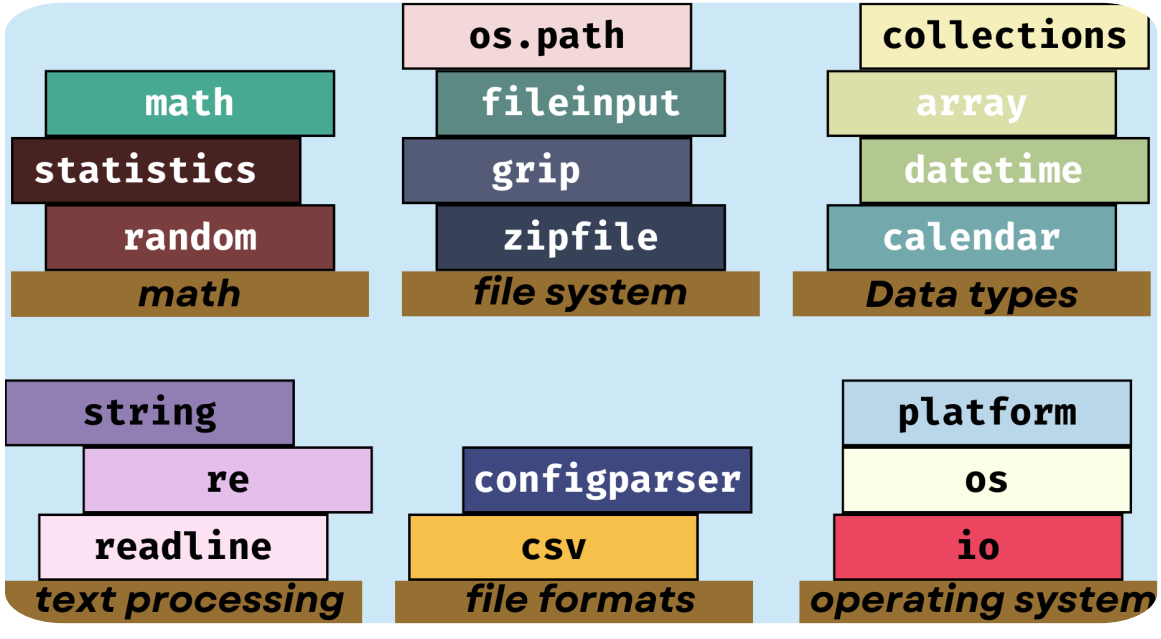
Standardní knihovna je obrovská:



- práce se soubory
- JSON, CSV, XML
- HTTP server
- regulární výrazy
- multiprocessing
- testování

PYTHON MÁ „BATERIE V BALENÍ“

Standardní knihovna je obrovská:



- práce se soubory
- JSON, CSV, XML
- HTTP server
- regulární výrazy
- multiprocessing
- testování

import a hotovo

😉 PYTHON JE EXTRÉMNĚ TOLERANTNÍ (A TO JE PROBLÉM)

😉 PYTHON JE EXTRÉMNĚ TOLERANTNÍ (A TO JE PROBLÉM)

- netrestá hned

😉 PYTHON JE EXTRÉMNĚ TOLERANTNÍ (A TO JE PROBLÉM)

- netrestá hned
- dovolí ti spoustu věcí

😉 PYTHON JE EXTRÉMNĚ TOLERANTNÍ (A TO JE PROBLÉM)

- netrestá hned
- dovolí ti spoustu věcí
- chyby často najdeš až za běhu

😉 PYTHON JE EXTRÉMNĚ TOLERANTNÍ (A TO JE PROBLÉM)

- netrestá hned
- dovolí ti spoustu věcí
- chyby často najdeš až za běhu

```
1 x = 5 # int 😊
2 x = "hello" # string 😳
```

😉 PYTHON JE EXTRÉMNĚ TOLERANTNÍ (A TO JE PROBLÉM)

- netrestá hned
- dovolí ti spoustu věcí
- chyby často najdeš až za běhu

```
1 x = 5 # int 😊
2 x = "hello" # string 😳
```

😉 PYTHON JE EXTRÉMNĚ TOLERANTNÍ (A TO JE PROBLÉM)

- netrestá hned
- dovolí ti spoustu věcí
- chyby často najdeš až za běhu

```
1 x = 5 # int 😊
2 x = "hello" # string 😳
```

😉 PYTHON JE EXTRÉMNĚ TOLERANTNÍ (A TO JE PROBLÉM)

- netrestá hned
- dovolí ti spoustu věcí
- chyby často najdeš až za běhu

```
1  x = 5 # int 😊
2  x = "hello" # string 🥴
```

✅ Skvělé pro začátek

😉 PYTHON JE EXTRÉMNĚ TOLERANTNÍ (A TO JE PROBLÉM)

- netrestá hned
- dovolí ti spoustu věcí
- chyby často najdeš až za běhu

```
1  x = 5 # int 😊
2  x = "hello" # string 🥴
```

✅ Skvělé pro začátek

💣 Nebezpečné ve velkých projektech

❌ KDE PYTHON NENÍ VHODNÝ

❌ KDE PYTHON NENÍ VHODNÝ

- Real-time systémy

❌ KDE PYTHON NENÍ VHODNÝ

- Real-time systémy
- Embedded / mikrokontroléry bez OS

❌ KDE PYTHON NENÍ VHODNÝ

- Real-time systémy
- Embedded / mikrokontroléry bez OS
- Extrémně výkonnostní smyčky

❌ KDE PYTHON NENÍ VHODNÝ

- Real-time systémy
- Embedded / mikrokontroléry bez OS
- Extrémně výkonnostní smyčky
- Aplikace s tvrdými časovými limity

❌ KDE PYTHON NENÍ VHODNÝ

- Real-time systémy
- Embedded / mikrokontroléry bez OS
- Extrémně výkonnostní smyčky
- Aplikace s tvrdými časovými limity
- Nízká spotřeba paměti / energie

❌ KDE PYTHON NENÍ VHODNÝ

- Real-time systémy
- Embedded / mikrokontroléry bez OS
- Extrémně výkonnostní smyčky
- Aplikace s tvrdými časovými limity
- Nízká spotřeba paměti / energie

“Python není špatný jazyk.
Je to špatný jazyk na špatné problémy.”

🌐 EKOSYSTÉM & POUŽITÍ PYTHONU

🌐 EKOSYSTÉM & POUŽITÍ PYTHONU

🔬 Věda & výzkum ➡️ Matplotlib, OpenCV, wxPython

🌐 EKOSYSTÉM & POUŽITÍ PYTHONU

🔬 Věda & výzkum ➡️ Matplotlib, OpenCV, wxPython

⚙️ Automatizace & skripty ➡️ buildy, deployment, práce se soubory

🌐 EKOSYSTÉM & POUŽITÍ PYTHONU

🔬 Věda & výzkum ➡️ Matplotlib, OpenCV, wxPython

⚙️ Automatizace & skripty ➡️ buildy, deployment, práce se soubory

🕸️ Web ➡️ Django, Flask, FastAPI

🌐 EKOSYSTÉM & POUŽITÍ PYTHONU

🔬 Věda & výzkum ➡️ Matplotlib, OpenCV, wxPython

⚙️ Automatizace & skripty ➡️ buildy, deployment, práce se soubory

🕸️ Web ➡️ Django, Flask, FastAPI

📊 Data & AI ➡️ NumPy, Pandas, TensorFlow, PyTorch

🌐 EKOSYSTÉM & POUŽITÍ PYTHONU

🔬 Věda & výzkum ➡️ Matplotlib, OpenCV, wxPython

⚙️ Automatizace & skripty ➡️ buildy, deployment, práce se soubory

🕸️ Web ➡️ Django, Flask, FastAPI

📊 Data & AI ➡️ NumPy, Pandas, TensorFlow, PyTorch

🧪 Testování ➡️ pytest, unittest

🌐 EKOSYSTÉM & POUŽITÍ PYTHONU

🔬 Věda & výzkum ➡️ Matplotlib, OpenCV, wxPython

⚙️ Automatizace & skripty ➡️ buildy, deployment, práce se soubory

🕸️ Web ➡️ Django, Flask, FastAPI

📊 Data & AI ➡️ NumPy, Pandas, TensorFlow, PyTorch

🧪 Testování ➡️ pytest, unittest

“Python není jeden obor.
🔨 Python je nástrojový jazyk.”

PORTABILITA PYTHONU

PORTABILITA PYTHONU

Python kód je přenositelný mezi platformami.

💻 PORTABILITA PYTHONU 🖥️

Python kód je přenositelný mezi platformami.

Stejný skript běží na:

💻 PORTABILITA PYTHONU 🖥️

Python kód je přenositelný mezi platformami.

Stejný skript běží na:

- Windows

💻 PORTABILITA PYTHONU 🖥️

Python kód je přenositelný mezi platformami.

Stejný skript běží na:

- Windows
- Linux

💻 PORTABILITA PYTHONU 🖥️

Python kód je přenositelný mezi platformami.

Stejný skript běží na:

- Windows
- Linux
- macOS

💻 PORTABILITA PYTHONU 🖥️

Python kód je přenositelný mezi platformami.

Stejný skript běží na:

- Windows
- Linux
- macOS

```
1 print("Hello world")
```

💻 PORTABILITA PYTHONU 🖥️

Python kód je přenositelný mezi platformami.

Stejný skript běží na:

- Windows
- Linux
- macOS

```
1 print("Hello world")
```

➡️ bez rekompilace, bez změny kódu*

✍️ JAK SE PYTHON TYPICKY PÍŠE

✍️ JAK SE PYTHON TYPICKY PÍŠE

✅ PYTHON SE TYPICKY PÍŠE JAKO:

✍️ JAK SE PYTHON TYPICKY PÍŠE

✅ PYTHON SE TYPICKY PÍŠE JAKO:

- malé skripty

✍️ JAK SE PYTHON TYPICKY PÍŠE

✅ PYTHON SE TYPICKY PÍŠE JAKO:

- malé skripty
- moduly, které se importují

✍️ JAK SE PYTHON TYPICKY PÍŠE

✅ PYTHON SE TYPICKY PÍŠE JAKO:

- malé skripty
- moduly, které se importují
- krátké funkce

✍️ JAK SE PYTHON TYPICKY PÍŠE

✅ PYTHON SE TYPICKY PÍŠE JAKO:

- malé skripty
- moduly, které se importují
- krátké funkce
- kód, který se často spouští a upravuje

✍️ JAK SE PYTHON TYPICKY PÍŠE

✅ PYTHON SE TYPICKY PÍŠE JAKO:

- malé skripty
- moduly, které se importují
- krátké funkce
- kód, který se často spouští a upravuje

❌ NE JAKO:

✍️ JAK SE PYTHON TYPICKY PÍŠE

✅ PYTHON SE TYPICKY PÍŠE JAKO:

- malé skripty
- moduly, které se importují
- krátké funkce
- kód, který se často spouští a upravuje

❌ NE JAKO:

- jeden obří program

✍️ JAK SE PYTHON TYPICKY PÍŠE

✅ PYTHON SE TYPICKY PÍŠE JAKO:

- malé skripty
- moduly, které se importují
- krátké funkce
- kód, který se často spouští a upravuje

❌ NE JAKO:

- jeden obří program
- tisíce řádků v jednom souboru

✍️ JAK SE PYTHON TYPICKY PÍŠE

✅ PYTHON SE TYPICKY PÍŠE JAKO:

- malé skripty
- moduly, které se importují
- krátké funkce
- kód, který se často spouští a upravuje

❌ NE JAKO:

- jeden obří program
- tisíce řádků v jednom souboru
- build krok mezi změnou a spuštěním programu

✍️ JAK SE PYTHON TYPICKY PÍŠE

✅ PYTHON SE TYPICKY PÍŠE JAKO:

- malé skripty
- moduly, které se importují
- krátké funkce
- kód, který se často spouští a upravuje

❌ NE JAKO:

- jeden obří program
- tisíce řádků v jednom souboru
- build krok mezi změnou a spuštěním programu

“Python je jazyk pro přemýšlení u problému, ne o jazyku.”

🚫 CO SE MI NELÍBÍ V PYTHONU

🚫 CO SE MI NELÍBÍ V PYTHONU

- ❌ Neexistuje **skutečná const proměnná**

🚫 CO SE MI NELÍBÍ V PYTHONU

- ❌ Neexistuje **skutečná const proměnná**
- ❌ Omezený **scope** (žádné libovolné bloky pomocí { }

🚫 CO SE MI NELÍBÍ V PYTHONU

- ❌ Neexistuje **skutečná const proměnná**
- ❌ Omezený **scope** (žádné libovolné bloky pomocí { }

Python je jednoduchý, ale platí za to menší kontrolou.

❌ CONST PROMĚNNÉ V PYTHONU

❌ CONST PROMĚNNÉ V PYTHONU

```
1 PI = 3.14 # float 3.14
2 PI = 20   # funguje bez chyby
```

❌ CONST PROMĚNNÉ V PYTHONU

```
1 PI = 3.14 # float 3.14
2 PI = 20   # funguje bez chyby
```

Velká písmena jsou jen **konvence**, ne jazykové pravidlo.

SCOPE (ROZSAH PLATNOSTI)

SCOPE (ROZSAH PLATNOSTI)

V Pythonu **neexistuje blokový scope** pomocí { }.

SCOPE (ROZSAH PLATNOSTI)

V Pythonu **neexistuje blokový scope** pomocí { }.

```
1 if True:
2   x = 10
3
4 print(x)  # x stále existuje
```

SCOPE (ROZSAH PLATNOSTI)

V Pythonu **neexistuje blokový scope** pomocí { }.

```
1 if True:
2   x = 10
3
4 print(x)  # x stále existuje
```

Scope vzniká pouze:

SCOPE (ROZSAH PLATNOSTI)

V Pythonu **neexistuje blokový scope** pomocí { }.

```
1 if True:
2   x = 10
3
4 print(x)  # x stále existuje
```

Scope vzniká pouze:

- funkcí (def)

SCOPE (ROZSAH PLATNOSTI)

V Pythonu **neexistuje blokový scope** pomocí { }.

```
1 if True:
2   x = 10
3
4 print(x)  # x stále existuje
```

Scope vzniká pouze:

- funkcí (`def`)
- třídou (`class`)

SCOPE (ROZSAH PLATNOSTI)

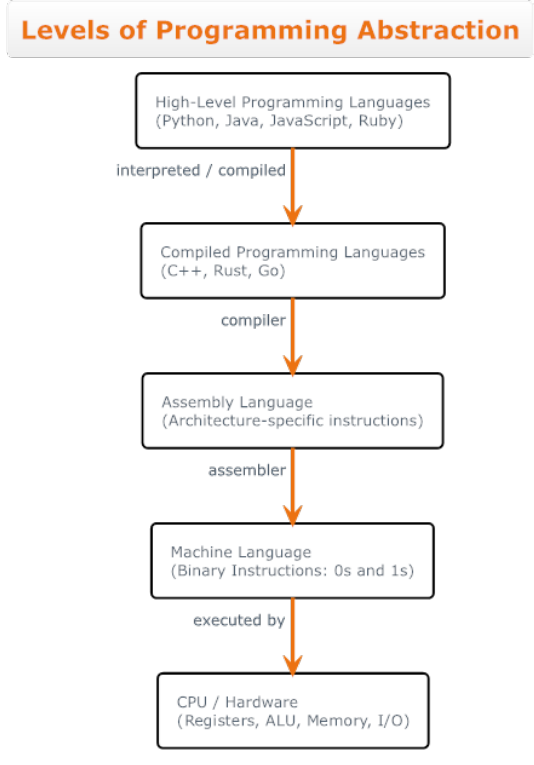
V Pythonu **neexistuje blokový scope** pomocí { }.

```
1 if True:
2   x = 10
3
4 print(x)  # x stále existuje
```

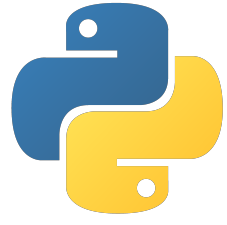
Scope vzniká pouze:

- funkcí (`def`)
- třídou (`class`)
- modulem (soubor)

PROGRAMOVACÍ JAZYKY



👑 INTERPRETOVANÉ PROGRAMOVACÍ JAZYKY



PYTHON

```python
total = 0

for i in range(1, 11): # [1;11)
  total += i # total = total + i

print(total) # 55
```

1. Deklarace proměnné
2. Smyčka for
3. Přičítání v těle smyčky (total = total + i)
4. Výpis výsledku

PYTHON

```python
total = 0

for i in range(1, 11): # [1;11)
  total += i # total = total + i

print(total) # 55
```

1. Deklarace proměnné
2. Smyčka for
3. Přičítání v těle smyčky (total = total + i)
4. Výpis výsledku

PYTHON

```python
total = 0

for i in range(1, 11): # [1;11)
  total += i # total = total + i

print(total) # 55
```

1. Deklarace proměnné
2. Smyčka for
3. Přičítání v těle smyčky (total = total + i)
4. Výpis výsledku

PYTHON

```python
total = 0

for i in range(1, 11): # [1;11)
  total += i # total = total + i

print(total) # 55
```

1. Deklarace proměnné
2. Smyčka for
3. Přičítání v těle smyčky (total = total + i)
4. Výpis výsledku

PYTHON

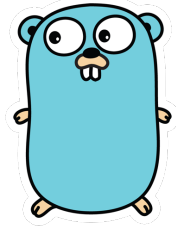
```python
total = 0

for i in range(1, 11): # [1;11)
  total += i # total = total + i

print(total) # 55
```

1. Deklarace proměnné
2. Smyčka for
3. Přičítání v těle smyčky (total = total + i)
4. Výpis výsledku

🏎️ KOMPILOVANÉ 🐈 🗡️ PROGRAMOVACÍ JAZYKY



C++

```cpp
#include <iostream>

int main()
{
  int total = 0;

  for (int i = 1; i <= 10; ++i)
    total += i; // total = total + i

  std::cout << total << std::endl; // 55
  return 0;
}
```

C++

```cpp
#include <iostream>

int main()
{
  int total = 0;

  for (int i = 1; i <= 10; ++i)
    total += i; // total = total + i

  std::cout << total << std::endl; // 55
  return 0;
}
```

C++

```cpp
#include <iostream>

int main()
{
  int total = 0;

  for (int i = 1; i <= 10; ++i)
    total += i; // total = total + i

  std::cout << total << std::endl; // 55
  return 0;
}
```

C++

```cpp
#include <iostream>

int main()
{
  int total = 0;

  for (int i = 1; i <= 10; ++i)
    total += i; // total = total + i

  std::cout << total << std::endl; // 55
  return 0;
}
```

C++

```cpp
#include <iostream>

int main()
{
  int total = 0;

  for (int i = 1; i <= 10; ++i)
    total += i; // total = total + i

  std::cout << total << std::endl; // 55
  return 0;
}
```

C++

```cpp
#include <iostream>

int main()
{
  int total = 0;

  for (int i = 1; i <= 10; ++i)
    total += i; // total = total + i

  std::cout << total << std::endl; // 55
  return 0;
}
```

C++

```cpp
#include <iostream>

int main()
{
  int total = 0;

  for (int i = 1; i <= 10; ++i)
    total += i; // total = total + i

  std::cout << total << std::endl; // 55
  return 0;
}
```

C++

```cpp
#include <iostream>

int main()
{
  int total = 0;

  for (int i = 1; i <= 10; ++i)
    total += i; // total = total + i

  std::cout << total << std::endl; // 55
  return 0;
}
```

C++

```cpp
#include <iostream>

int main()
{
  int total = 0;

  for (int i = 1; i <= 10; ++i)
    total += i; // total = total + i

  std::cout << total << std::endl; // 55
  return 0;
}
```

ASSEMBLY

Druh jazyka

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0       ; total = 0
 6    mov rcx, 1       ; i = 1
 7
 8  loop:
 9    add rax, rcx     ; total += i
10    inc rcx          ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax     ; exit code = total
15    mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
1  section .text
2    global _start
3
4  _start:
5    mov rax, 0       ; total = 0
6    mov rcx, 1       ; i = 1
7
8  loop:
9    add rax, rcx     ; total += i
10   inc rcx          ; i++
11   cmp rcx, 11
12   jne loop
13
14   mov rdi, rax     ; exit code = total
15   mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0       ; total = 0
 6    mov rcx, 1       ; i = 1
 7
 8  loop:
 9    add rax, rcx     ; total += i
10    inc rcx          ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax     ; exit code = total
15    mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0       ; total = 0
 6    mov rcx, 1       ; i = 1
 7
 8  loop:
 9    add rax, rcx     ; total += i
10    inc rcx          ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax     ; exit code = total
15    mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0        ; total = 0
 6    mov rcx, 1        ; i = 1
 7
 8  loop:
 9    add rax, rcx      ; total += i
10    inc rcx           ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax      ; exit code = total
15    mov rax, 60       ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
1  section .text
2    global _start
3
4  _start:
5    mov rax, 0       ; total = 0
6    mov rcx, 1       ; i = 1
7
8  loop:
9    add rax, rcx     ; total += i
10   inc rcx          ; i++
11   cmp rcx, 11
12   jne loop
13
14   mov rdi, rax     ; exit code = total
15   mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0       ; total = 0
 6    mov rcx, 1       ; i = 1
 7
 8  loop:
 9    add rax, rcx     ; total += i
10    inc rcx          ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax     ; exit code = total
15    mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0       ; total = 0
 6    mov rcx, 1       ; i = 1
 7
 8  loop:
 9    add rax, rcx     ; total += i
10    inc rcx          ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax     ; exit code = total
15    mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0       ; total = 0
 6    mov rcx, 1       ; i = 1
 7
 8  loop:
 9    add rax, rcx     ; total += i
10    inc rcx          ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax     ; exit code = total
15    mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0       ; total = 0
 6    mov rcx, 1       ; i = 1
 7
 8  loop:
 9    add rax, rcx     ; total += i
10    inc rcx          ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax     ; exit code = total
15    mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0       ; total = 0
 6    mov rcx, 1       ; i = 1
 7
 8  loop:
 9    add rax, rcx     ; total += i
10    inc rcx          ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax     ; exit code = total
15    mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0       ; total = 0
 6    mov rcx, 1       ; i = 1
 7
 8  loop:
 9    add rax, rcx     ; total += i
10    inc rcx          ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax     ; exit code = total
15    mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0       ; total = 0
 6    mov rcx, 1       ; i = 1
 7
 8  loop:
 9    add rax, rcx     ; total += i
10    inc rcx          ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax     ; exit code = total
15    mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0       ; total = 0
 6    mov rcx, 1       ; i = 1
 7
 8  loop:
 9    add rax, rcx     ; total += i
10    inc rcx          ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax     ; exit code = total
15    mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

ASSEMBLY

```
 1  section .text
 2    global _start
 3
 4  _start:
 5    mov rax, 0       ; total = 0
 6    mov rcx, 1       ; i = 1
 7
 8  loop:
 9    add rax, rcx     ; total += i
10    inc rcx          ; i++
11    cmp rcx, 11
12    jne loop
13
14    mov rdi, rax     ; exit code = total
15    mov rax, 60      ; sys_exit
```

Každá instrukce = jeden jasný krok procesoru

RYCHLOST

LEIBNIZ FORMULA FOR PI

Gottfried Wilhelm Leibniz

1646 - 1716

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1},$$

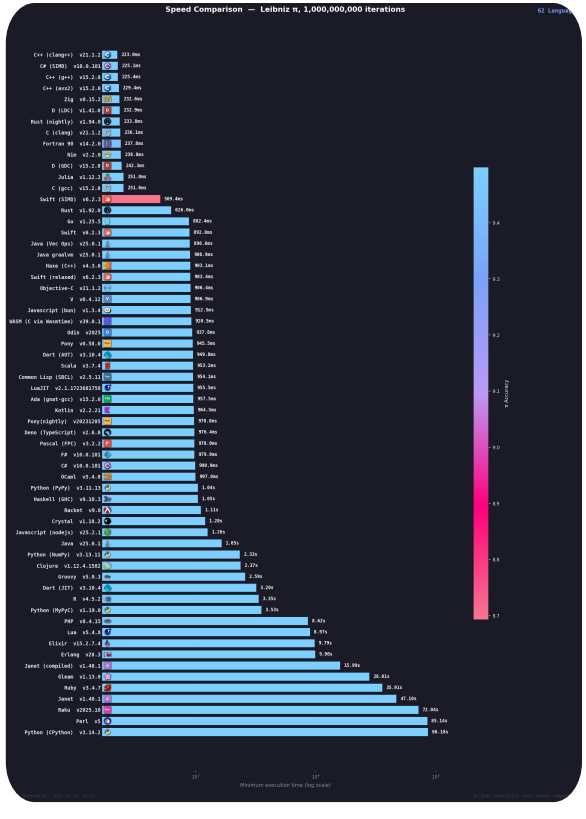
BENCHMARK RYCHLOSTÍ PROGRAMOVACÍCH JAZYKŮ

1'000'000'000 iteraci (Speed Comparison)

BENCHMARK RYCHLOSTÍ PROGRAMOVACÍCH JAZYKŮ

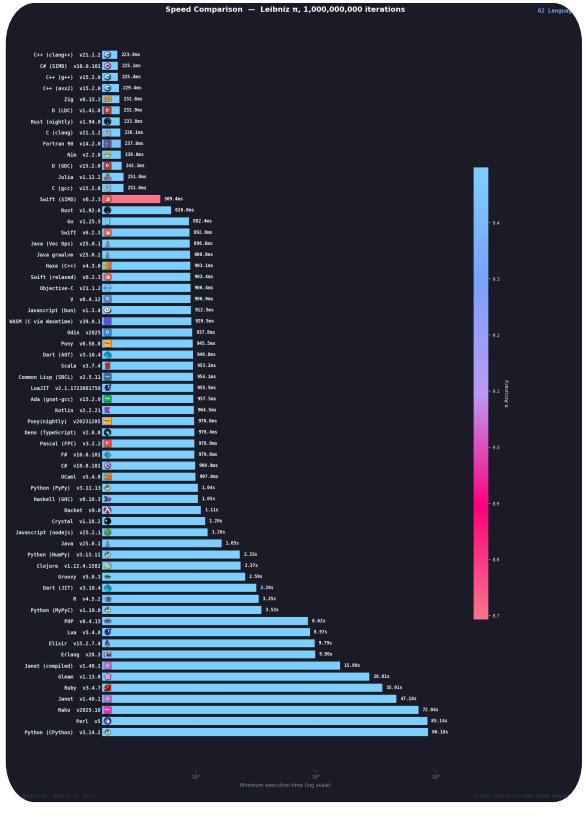
1'000'000'000 iteraci (Speed Comparison)

BENCHMARK RYCHLOSTÍ PROGRAMOVACÍCH JAZYKŮ



1'000'000'000 iteraci (Speed Comparison)

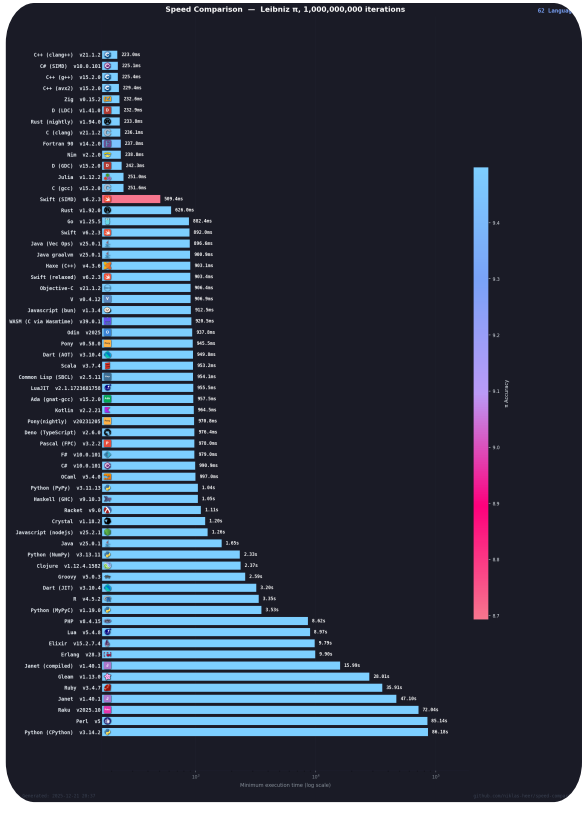
BENCHMARK RYCHLOSTÍ PROGRAMOVACÍCH JAZYKŮ



C++ (clang++)	v21.1.2	223.0ms
C# (SIMD)	v10.0.101	225.1ms
C++ (g++)	v15.2.0	225.4ms
C++ (avx2)	v15.2.0	229.4ms

1'000'000'000 iteraci (Speed Comparison)

BENCHMARK RYCHLOSTÍ PROGRAMOVACÍCH JAZYKŮ

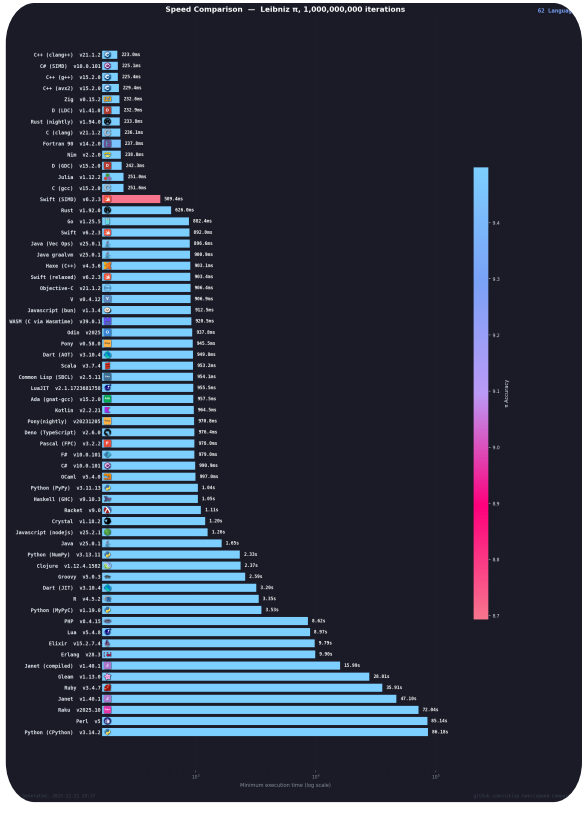


C++ (clang++)	v21.1.2	223.0ms
C# (SIMD)	v10.0.101	225.1ms
C++ (g++)	v15.2.0	225.4ms
C++ (avx2)	v15.2.0	229.4ms

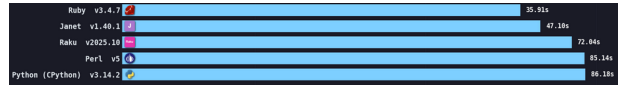
0.223 [s]

1'000'000'000 iteraci (Speed Comparison)

BENCHMARK RYCHLOSTÍ PROGRAMOVACÍCH JAZYKŮ

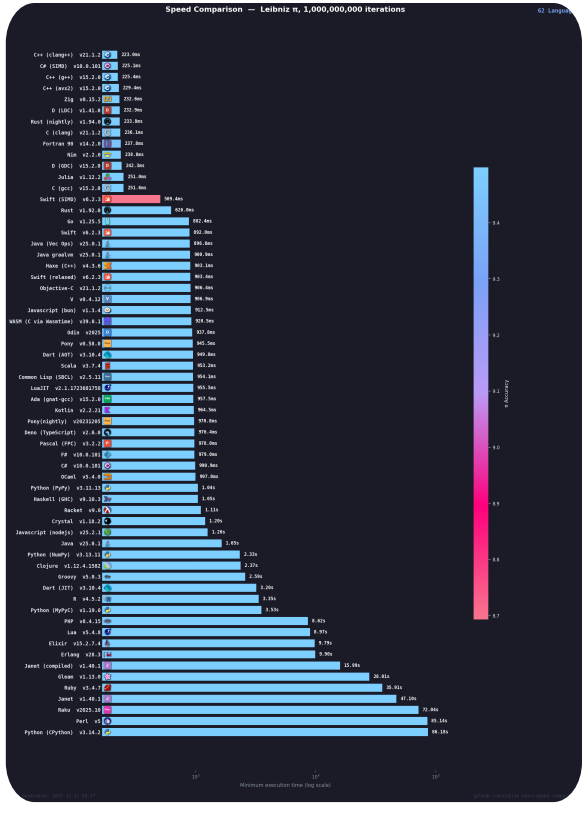


0.223 [s]

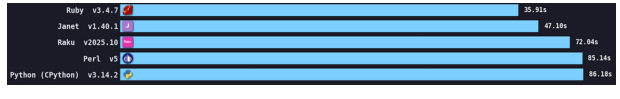


1'000'000'000 iteraci (Speed Comparison)

BENCHMARK RYCHLOSTÍ PROGRAMOVACÍCH JAZYKŮ



0.223 [s]



86.18 [s]

1'000'000'000 iteraci (Speed Comparison)

PYTHON JE ~386X POMALEJŠÍ

PROČ?

PROČ?

PROČ?

PROČ?

KOMPILOVANÝ KÓD

KOMPILOVANÝ KÓD

KOMPILOVANÝ KÓD

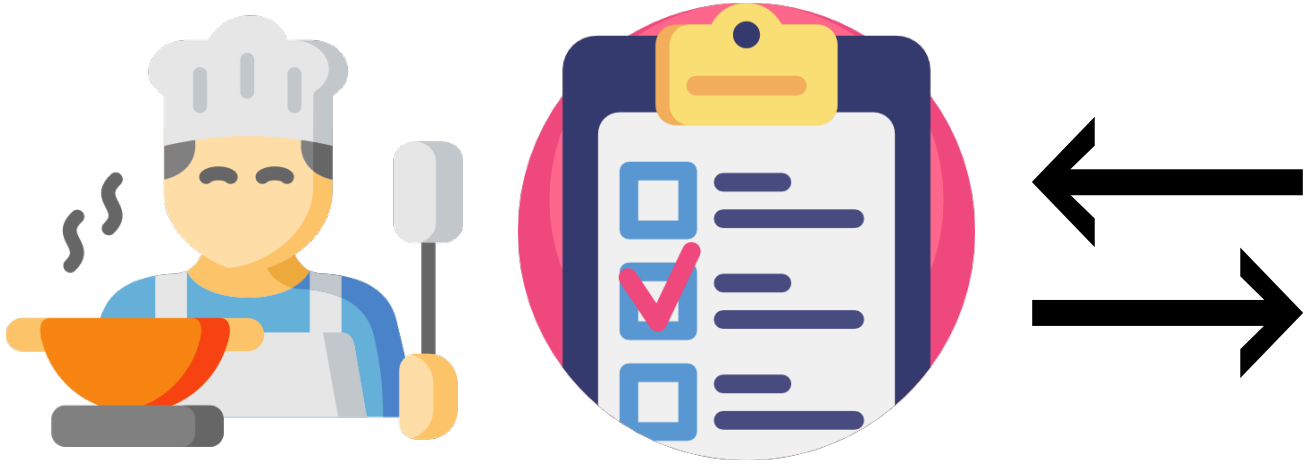
KOMPILOVANÝ KÓD

INTERPRETOVANÝ KÓD

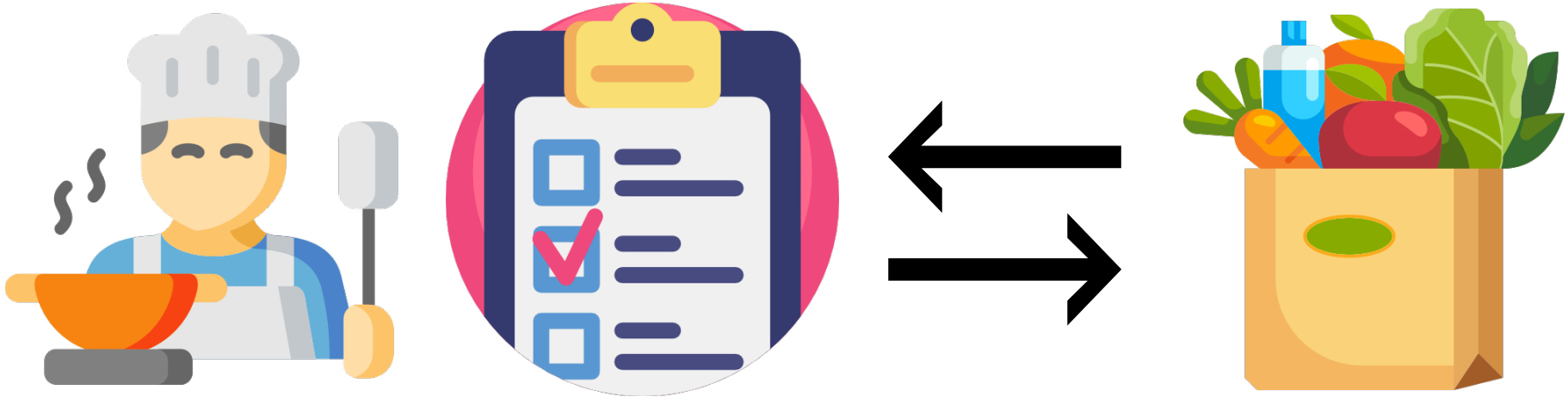
INTERPRETOVANÝ KÓD

INTERPRETOVANÝ KÓD

INTERPRETOVANÝ KÓD



INTERPRETOVANÝ KÓD



CO JE OBJEKT V PYTHONU

CO JE OBJEKT V PYTHONU

Objekt je konkrétní instance nějakého typu (třídy).

CO JE OBJEKT V PYTHONU

Objekt je konkrétní instance nějakého typu (třídy).

Jednoduše řečeno: **objekt = data + chování**

🧱 CO JE OBJEKT V PYTHONU

Objekt je konkrétní instance nějakého typu (třídy).

Jednoduše řečeno: **objekt = data + chování**

```
1  x = 10 # int
```

🧱 CO JE OBJEKT V PYTHONU

Objekt je konkrétní instance nějakého typu (třídy).

Jednoduše řečeno: **objekt = data + chování**

```
1  x = 10 # int
```

Proměnná x není jen číslo – je to **objekt typu int**.

V PYTHONU JE VŠECHNO OBJEKT

V PYTHONU JE VŠECHNO OBJEKT

- Čísla (`int`, `float`)

V PYTHONU JE VŠECHNO OBJEKT

- Čísla (`int`, `float`)
- Řetězce (`str`)

V PYTHONU JE VŠECHNO OBJEKT

- Čísla (`int`, `float`)
- Řetězce (`str`)
- Seznamy (`list`)

V PYTHONU JE VŠECHNO OBJEKT

- Čísla (`int`, `float`)
- Řetězce (`str`)
- Seznamy (`list`)
- Funkce

V PYTHONU JE VŠECHNO OBJEKT

- Čísla (`int`, `float`)
- Řetězce (`str`)
- Seznamy (`list`)
- Funkce
- Třídy

V PYTHONU JE VŠECHNO OBJEKT

- Čísla (`int`, `float`)
- Řetězce (`str`)
- Seznamy (`list`)
- Funkce
- Třídy

```
1 print(type(10)) # class 'int'
2 print(type("ahoj")) # class 'str'
3 print(type(len)) # class 'builtin_function_or_method'
```

V PYTHONU JE VŠECHNO OBJEKT

- Čísla (`int`, `float`)
- Řetězce (`str`)
- Seznamy (`list`)
- Funkce
- Třídy

```
1 print(type(10)) # class 'int'
2 print(type("ahoj")) # class 'str'
3 print(type(len)) # class 'builtin_function_or_method'
```

V PYTHONU JE VŠECHNO OBJEKT

- Čísla (`int`, `float`)
- Řetězce (`str`)
- Seznamy (`list`)
- Funkce
- Třídy

```
1 print(type(10)) # class 'int'
2 print(type("ahoj")) # class 'str'
3 print(type(len)) # class 'builtin_function_or_method'
```

🧠 V PYTHONU JE VŠECHNO OBJEKT

- Čísla (`int`, `float`)
- Řetězce (`str`)
- Seznamy (`list`)
- Funkce
- Třídy

```
1 print(type(10)) # class 'int'
2 print(type("ahoj")) # class 'str'
3 print(type(len)) # class 'builtin_function_or_method'
```

VLASTNOSTI OBJEKTU

VLASTNOSTI OBJEKTU

Objekt má **vlastnosti**, kterým říkáme **atributy**.

VLASTNOSTI OBJEKTU

Objekt má **vlastnosti**, kterým říkáme **atributy**.

```
1 class Employee:
2   def __init__(self, name='Radovan', salary=18000):
3     self.name = name # atribut
4     self.salary = salary #atribut
5
6   def show(self): # metoda
7     print(self.name)
8     print(self.salary)
```

VLASTNOSTI OBJEKTU

Objekt má **vlastnosti**, kterým říkáme **atributy**.

```
1 class Employee:
2   def __init__(self, name='Radovan', salary=18000):
3     self.name = name # atribut
4     self.salary = salary #atribut
5
6   def show(self): # metoda
7     print(self.name)
8     print(self.salary)
```

VLASTNOSTI OBJEKTU

Objekt má **vlastnosti**, kterým říkáme **atributy**.

```
1 class Employee:
2   def __init__(self, name='Radovan', salary=18000):
3     self.name = name # atribut
4     self.salary = salary #atribut
5
6   def show(self): # metoda
7     print(self.name)
8     print(self.salary)
```

VLASTNOSTI OBJEKTU

Objekt má **vlastnosti**, kterým říkáme **atributy**.

```
1 class Employee:
2   def __init__(self, name='Radovan', salary=18000):
3     self.name = name # atribut
4     self.salary = salary #atribut
5
6   def show(self): # metoda
7     print(self.name)
8     print(self.salary)
```

VLASTNOSTI OBJEKTU

Objekt má **vlastnosti**, kterým říkáme **atributy**.

```
1 class Employee:
2   def __init__(self, name='Radovan', salary=18000):
3     self.name = name # atribut
4     self.salary = salary #atribut
5
6   def show(self): # metoda
7     print(self.name)
8     print(self.salary)
```

VLASTNOSTI OBJEKTU

Objekt má **vlastnosti**, kterým říkáme **atributy**.

```
1 class Employee:
2   def __init__(self, name='Radovan', salary=18000):
3     self.name = name # atribut
4     self.salary = salary #atribut
5
6   def show(self): # metoda
7     print(self.name)
8     print(self.salary)
```

Atributy popisují **stav** objektu.

METODY OBJEKTU

METODY OBJEKTU

Metody jsou funkce, které patří objektu.

🔧 METODY OBJEKTU

Metody jsou funkce, které patří objektu.

```
1 text = "python" #str
2 print(text.upper()) # PYTHON
3 print(text.count("o")) # 1
```

🔧 METODY OBJEKTU

Metody jsou funkce, které patří objektu.

```
1 text = "python" #str
2 print(text.upper()) # PYTHON
3 print(text.count("o")) # 1
```

🔧 METODY OBJEKTU

Metody jsou funkce, které patří objektu.

```
1 text = "python" #str
2 print(text.upper()) # PYTHON
3 print(text.count("o")) # 1
```

🔧 METODY OBJEKTU

Metody jsou funkce, které patří objektu.

```
1 text = "python" #str
2 print(text.upper()) # PYTHON
3 print(text.count("o")) # 1
```

🔧 METODY OBJEKTU

Metody jsou funkce, které patří objektu.

```
1 text = "python" #str
2 print(text.upper()) # PYTHON
3 print(text.count("o")) # 1
```

Metody popisují **chování** objektu.

🔧 METODY OBJEKTU

Metody jsou funkce, které patří objektu.

```
1 text = "python" #str
2 print(text.upper()) # PYTHON
3 print(text.count("o")) # 1
```

Metody popisují **chování** objektu.

Každý 'str' objekt má **45** metod.

🔍 JAK ZJISTIT, CO OBJEKT UMÍ

🔍 JAK ZJISTIT, CO OBJEKT UMÍ

```
1 text = "python" # str
2 print(len(dir(text))) # 81
```

🔍 JAK ZJISTIT, CO OBJEKT UMÍ

```
1 text = "python" # str
2 print(len(dir(text))) # 81
```

🔍 JAK ZJISTIT, CO OBJEKT UMÍ

```
1 text = "python" # str
2 print(len(dir(text))) # 81
```

🔍 JAK ZJISTIT, CO OBJEKT UMÍ

```
1 text = "python" # str
2 print(len(dir(text))) # 81
```

Funkce `dir()` vypíše všechny atributy a metody objektu.

🔍 JAK ZJISTIT, CO OBJEKT UMÍ

```
1 text = "python" # str
2 print(len(dir(text))) # 81
```

Funkce `dir()` vypíše všechny atributy a metody objektu.

Skvělý nástroj pro objevování a učení.

SHRNUTÍ

SHRNUTÍ

- Objekt = data + chování

SHRNUTÍ

- Objekt = data + chování
- V Pythonu je všechno objekt

SHRNUTÍ

- Objekt = data + chování
- V Pythonu je všechno objekt
- Atributy = vlastnosti objektu

SHRNUTÍ

- Objekt = data + chování
- V Pythonu je všechno objekt
- Atributy = vlastnosti objektu
- Metody = co objekt umí dělat

🧩 SHRNUTÍ

- Objekt = data + chování
- V Pythonu je všechno objekt
- Atributy = vlastnosti objektu
- Metody = co objekt umí dělat

Python je objektový jazyk – a nesnaží se to skrývat.