

BUILT-IN DATA TYPES V PYTHONU

Co Python skutečně nabízí - bez obalu

Andrej Pčelovodov

PROČ ŘEŠIT DATOVÉ TYPY?

Andrej Pčelovodov

PROČ ŘEŠIT DATOVÉ TYPY?

-  Python je dynamicky typovaný

PROČ ŘEŠIT DATOVÉ TYPY?

-  Python je dynamicky typovaný
-  To neznamená „bez typů“

PROČ ŘEŠIT DATOVÉ TYPY?

-  Python je dynamicky typovaný
-  To neznamená „bez typů“
-    Typ určuje chování hodnoty



ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI

Počítač nerozumí slovům.
Rozumí jen dvěma stavům.



ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI

Andrej Pčelovodov

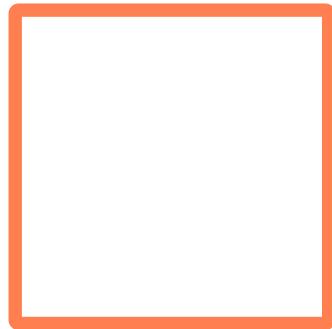


ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI

Andrej Pčelovodov



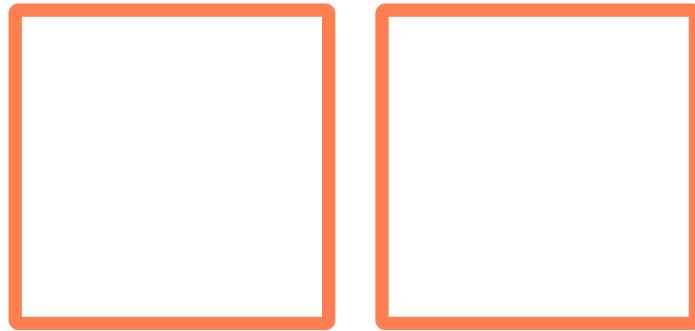
ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI



Andrej Pčelovodov



ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI



Andrej Pčelovodov



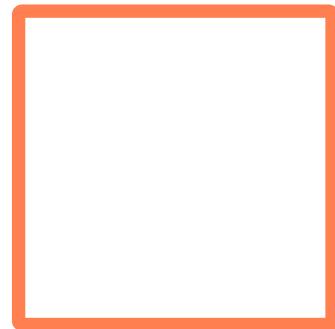
ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI



Andrej Pčelovodov



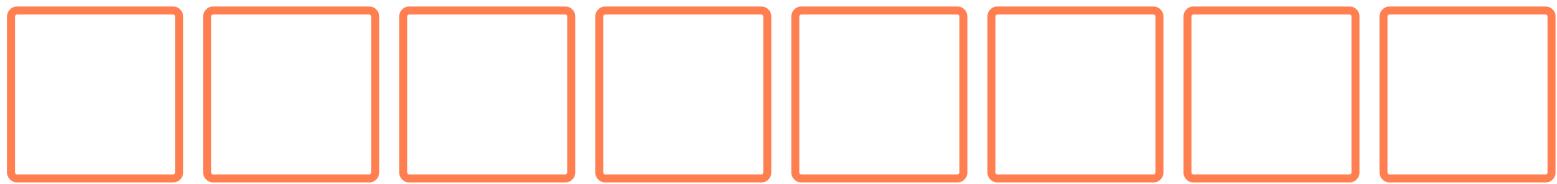
ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI



1 bit = nejmenší jednotka informace



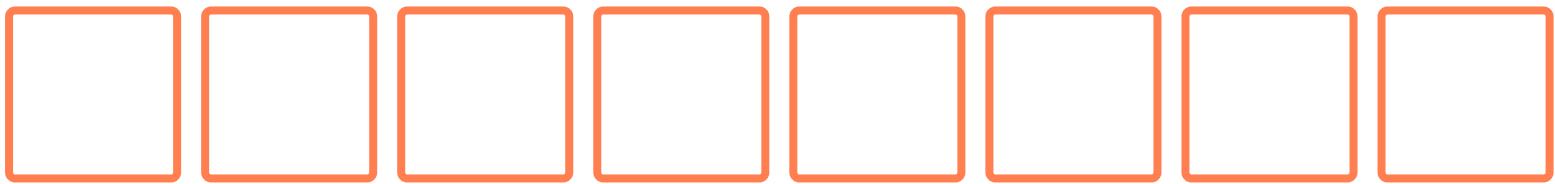
ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI



Andrej Pčelovodov



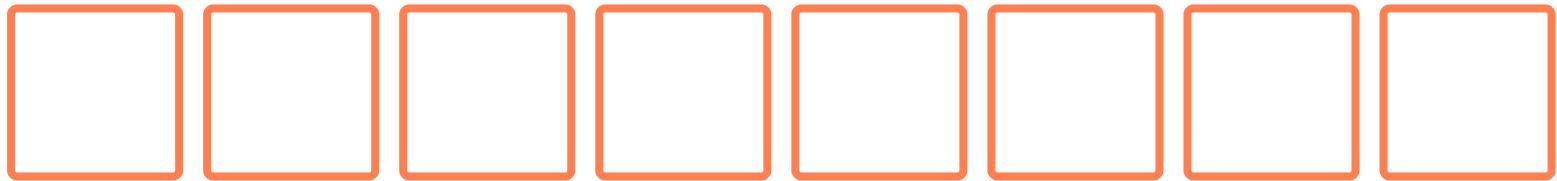
ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI



8 bitů = 1 byte



ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI



8 bitů = 1 byte

Kolik kombinací dostaneme?



ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI



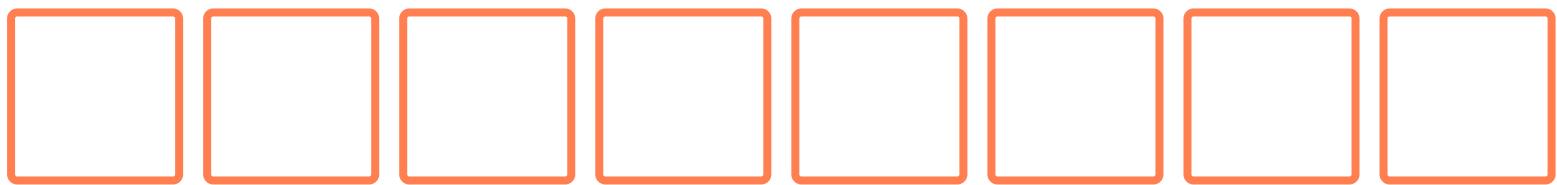
8 bitů = 1 byte

Kolik kombinací dostaneme?

$2^8 = 256$ hodnot



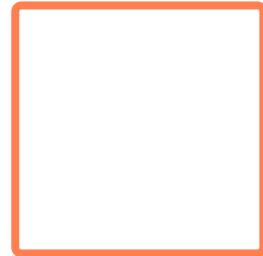
ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI



Andrej Pčelovodov



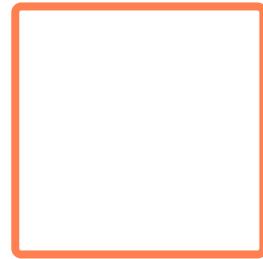
ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI



Andrej Pčelovodov



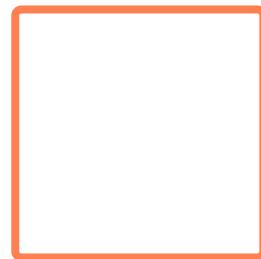
ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI



1 byte



ÚPLNÝ ZÁKLAD: INFORMACE V POČÍTAČI



1 byte

$$16 \text{ GB} = 16 * 1024 \text{ MB} = 16'384 * 1024 \text{ kB} = 16'777'216 * 1024 = \\ 17'179'869'184 \text{ byte}$$



A KDE JE V TOM BOOL?

Boolean je jen logická interpretace bitu.



A KDE JE V TOM BOOL?

- 1 **False** → 0
- 2 **True** → 1



BOOL V PYTHONU

```
1 a = True
2 b = False
3
4 print(int(a)) # 1
5 print(int(b)) # 0
```

Andrej Pčelovodov



BOOL JE VE SKUTEČNOSTI ČÍSLO

1 True + True

Andrej Pčelovodov



BOOL JE VE SKUTEČNOSTI ČÍSLO

```
1 True + True # 2
2 True * 10 # 10
```



SHRNUTÍ

- bit → 0 nebo 1
- 8 bitů → byte
- bool → interpretace 1 bitu jako True/False
- uvnitř je to pořád jen číslo



ZAPAMATUJ SI

Všechno v počítači je nakonec **jen hromada bitů**.
Datové typy jsou jen dohoda, jak ty bity číst.

1
2
3
4

ČÍSELNÉ TYPY (IMMUTABLE)

- int
- float
- complex
- fractions

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro leží čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro lepší čitelnost
```

int - CELÉ ČÍSLO

```
1 a = 10
2 print(type(a)) # class, 'int'
3
4 b = 4
5 print(type(b)) # class, 'int'
6
7 a + b    # 14      addition
8 a - b    # 6       subtraction
9 a * b    # 40     multiplication
10 a / b   # 2.5    true division
11 a // b  # 2      integer division
12 a % b   # 2      modulo operation (remainder of division)
13 a ** b  # 10_000 power operation
14
15 c = 10_000_000 #      - ien pro ležíčí čitelnost
```



V Pythonu nemá int limit velikosti - $[-\infty; \infty]$

Andrej Pčelovodov

int - CELÉ ČÍSLO

ZÁPORNÁ ČÍSLA

```
1 7 / 4 # 1.75 true division
2 7 // 4 # 1 integer division
3 -7 / 4 # -1.75 true division
4 -7 // 4 # -2 integer division
```

int - CELÉ ČÍSLO

ZÁPORNÁ ČÍSLA

```
1 7 / 4 # 1.75 true division
2 7 // 4 # 1 integer division
3 -7 / 4 # -1.75 true division
4 -7 // 4 # -2 integer division
```

int - CELÉ ČÍSLO

ZÁPORNÁ ČÍSLA

```
1 7 / 4 # 1.75 true division
2 7 // 4 # 1 integer division
3 -7 / 4 # -1.75 true division
4 -7 // 4 # -2 integer division
```

int - CELÉ ČÍSLO

ZÁPORNÁ ČÍSLA

```
1 7 / 4 # 1.75 true division
2 7 // 4 # 1 integer division
3 -7 / 4 # -1.75 true division
4 -7 // 4 # -2 integer division
```

int - CELÉ ČÍSLO

ZÁPORNÁ ČÍSLA

```
1 7 / 4 # 1.75 true division
2 7 // 4 # 1 integer division
3 -7 / 4 # -1.75 true division
4 -7 // 4 # -2 integer division
```

int - CELÉ ČÍSLO

ZÁPORNÁ ČÍSLA

```
1 7 / 4 # 1.75 true division
2 7 // 4 # 1 integer division
3 -7 / 4 # -1.75 true division
4 -7 // 4 # -2 integer division
```

📌 V Pythonu se celočíselné dělení (//) zaokrouhuje dolů, tj. k minus nekonečnu.

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True # True
9 False or True # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

boolean - PRAVDA / NEPRAVDA

```
1 int(True)      # 1
2 int(False)     # 0
3 bool(1)        # evaluates to True in a Boolean context
4 bool(-42)      # any non-zero number evaluates to True!
5 bool(0)        # False
6 not True       # False
7 not False      # True
8 True and True  # True
9 False or True  # True
10
11 1 + True      # 2
12 False + 42    # 42
13 7 - True      # 6
```

📌 Upcasting je operace převodu typů, která probíhá z podtřídy do nadřazené třídy. V tomto příkladu jsou hodnoty True a False, které patří do třídy odvozené od třídy integer, v případě potřeby převedeny zpět na celá čísla.

float - DESETINNÉ ČÍSLO

```
1 pi = 3.141592
2 radius = 4.5
3 print(type(radius)) # class 'float'
4 area = pi * (radius ** 2)
5 print(area) # 63.617238
```

float - DESETINNÉ ČÍSLO

```
1 pi = 3.141592
2 radius = 4.5
3 print(type(radius)) # class 'float'
4 area = pi * (radius ** 2)
5 print(area) # 63.617238
```

float - DESETINNÉ ČÍSLO

```
1 pi = 3.141592
2 radius = 4.5
3 print(type(radius)) # class 'float'
4 area = pi * (radius ** 2)
5 print(area) # 63.617238
```

float - DESETINNÉ ČÍSLO

```
1 pi = 3.141592
2 radius = 4.5
3 print(type(radius)) # class 'float'
4 area = pi * (radius ** 2)
5 print(area) # 63.617238
```

float - DESETINNÉ ČÍSLO

```
1 pi = 3.141592
2 radius = 4.5
3 print(type(radius)) # class 'float'
4 area = pi * (radius ** 2)
5 print(area) # 63.617238
```

float - DESETINNÉ ČÍSLO

```
1 pi = 3.141592
2 radius = 4.5
3 print(type(radius)) # class 'float'
4 area = pi * (radius ** 2)
5 print(area) # 63.617238
```

float - DESETINNÉ ČÍSLO

```
1 pi = 3.141592
2 radius = 4.5
3 print(type(radius)) # class 'float'
4 area = pi * (radius ** 2)
5 print(area) # 63.617238
```

! Python podporuje pouze double (64-bit) format.

float - DESETINNÉ ČÍSLO

```
1 import sys
2 sys.float_info
3 # sys.float_info
4 # (
5 #     max=1.7976931348623157e+308,
6 #     max_exp=1024,
7 #     max_10_exp=308,
8 #     min=2.2250738585072014e-308,
9 #     min_exp=-1021,
10 #     min_10_exp=-307,
11 #     dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2,
12 # )
```

float - DESETINNÉ ČÍSLO

```
1 import sys
2 sys.float_info
3 # sys.float_info
4 # (
5 #     max=1.7976931348623157e+308,
6 #     max_exp=1024,
7 #     max_10_exp=308,
8 #     min=2.2250738585072014e-308,
9 #     min_exp=-1021,
10 #     min_10_exp=-307,
11 #     dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2,
12 # )
```

float - DESETINNÉ ČÍSLO

```
1 import sys
2 sys.float_info
3 # sys.float_info
4 # (
5 #     max=1.7976931348623157e+308,
6 #     max_exp=1024,
7 #     max_10_exp=308,
8 #     min=2.2250738585072014e-308,
9 #     min_exp=-1021,
10 #     min_10_exp=-307,
11 #     dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2,
12 # )
```

float - DESETINNÉ ČÍSLO

```
1 import sys
2 sys.float_info
3 # sys.float_info
4 # (
5 #     max=1.7976931348623157e+308,
6 #     max_exp=1024,
7 #     max_10_exp=308,
8 #     min=2.2250738585072014e-308,
9 #     min_exp=-1021,
10 #     min_10_exp=-307,
11 #     dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2,
12 # )
```

float - DESETINNÉ ČÍSLO

```
1 import sys
2 sys.float_info
3 # sys.float_info
4 # (
5 #     max=1.7976931348623157e+308,
6 #     max_exp=1024,
7 #     max_10_exp=308,
8 #     min=2.2250738585072014e-308,
9 #     min_exp=-1021,
10 #     min_10_exp=-307,
11 #     dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2,
12 # )
```

float - DESETINNÉ ČÍSLO

```
1 import sys
2 sys.float_info
3 # sys.float_info
4 # (
5 #     max=1.7976931348623157e+308,
6 #     max_exp=1024,
7 #     max_10_exp=308,
8 #     min=2.2250738585072014e-308,
9 #     min_exp=-1021,
10 #     min_10_exp=-307,
11 #     dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2,
12 # )
```

[-1.7976931348623157e+308;1.7976931348623157e+308]

Andrej Pčelovodov

float - DESETINNÉ ČÍSLO

```
1 0.3 - 0.1 * 3 # this should be 0!!!
2 # -5.551115123125783e-17
```

float - DESETINNÉ ČÍSLO

```
1 0.3 - 0.1 * 3 # this should be 0!!!
2 # -5.551115123125783e-17
```

float - DESETINNÉ ČÍSLO

```
1 0.3 - 0.1 * 3 # this should be 0!!!
2 # -5.551115123125783e-17
```

float - DESETINNÉ ČÍSLO

```
1 0.3 - 0.1 * 3 # this should be 0!!!
2 # -5.551115123125783e-17
```

float - DESETINNÉ ČÍSLO

```
1 0.3 - 0.1 * 3 # this should be 0!!!
2 # -5.551115123125783e-17
```

float - DESETINNÉ ČÍSLO

```
1 0.3 - 0.1 * 3 # this should be 0!!!
2 # -5.551115123125783e-17
```



Typ Decimal

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.14000000000000124344978758017532527446746826171875')
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.14000000000000124344978758017532527446746826171875')
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.14000000000000124344978758017532527446746826171875')
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.14000000000000124344978758017532527446746826171875'
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.14000000000000124344978758017532527446746826171875')
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.14000000000000124344978758017532527446746826171875')
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.14000000000000124344978758017532527446746826171875')
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na typ Decimal
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.140000000000000124344978758017532527446746826171875')
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.14000000000000124344978758017532527446746826171875')
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.14000000000000124344978758017532527446746826171875')
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.14000000000000124344978758017532527446746826171875')
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.14000000000000124344978758017532527446746826171875')
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na typ Decimal
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```

float - DESETINNÉ ČÍSLO

```
1 from decimal import Decimal as D # pro pohodlnost
2 D(3.14) # protože konverze probíhá z typu float na typ Decimal
3 # Decimal('3.14000000000000124344978758017532527446746826171875')
4 D("3.14") # ze stringu, OK
5 # Decimal('3.14')
6 D(0.1) * D(3) - D(0.3) # protože konverze probíhá z typu float na
7 # Decimal('2.775557561565156540423631668E-17')
8 D("0.1") * D(3) - D("0.3") # string, int, string, OK
9 # Decimal('0.0')
10
11 D("3.2").as_integer_ratio() # 3.2 == 32/10 == 16/5
12 # (16, 5)
```



Construct Decimal from string or int!

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.7599999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.7599999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```

complex - KOMPLEXNÍ ČÍSLA

```
1 z = 3.14 + 2.73j
2 z = complex(3.14, 2.73)
3 print(z.real) # 3.14
4 print(z.imag) # 2.73
5 z.conjugate() # A + Bj -> A - Bj
6 # (3.14-2.73j)
7 z * 2 # násobení
8 # (6.28+5.46j)
9 z ** 2 # exponent
10 # (2.2420000000000018+17.3328j)
11 x = 1 + 1j
12 z - x
13 # (2.14+1.759999999999998j)
```



Python má komplexní čísla přímo v jazyce.

Andrej Pčelovodov

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```

fractions - ZLOMKY

```
1 from fractions import Fraction
2 Fraction(10, 6)
3 # Fraction(5, 3)
4 Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
5 # Fraction(1, 1)
6 f = Fraction(10, 6)
7 f.numerator
8 # 5
9 f.denominator
10 # 3
11 f.as_integer_ratio()
12 # (5, 3)
13
14 Fraction(0.125)
15 # Fraction(1, 8)
```



Finance a Science v Pythonu.

Andrej Pčelovodov



IMPLICITNÍ TYPE CASTING

Python při výpočtech automaticky povyšuje typy,
aby nepřišel o informaci.



IMPLICITNÍ TYPE CASTING

Python při výpočtech automaticky povyšuje typy,
aby nepřišel o informaci.

Castování probíhá vždy směrem nahoru.



HIERARCHIE ČÍSELNÝCH TYPŮ

```
bool → int → float → complex
```



HIERARCHIE ČÍSELNÝCH TYPŮ

```
bool → int → float → complex
```

- bool je podtřída int



HIERARCHIE ČÍSELNÝCH TYPŮ

```
bool → int → float → complex
```

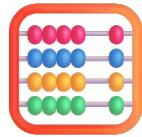
- `bool` je podtřída `int`
- `int` se povyšuje na `float`



HIERARCHIE ČÍSELNÝCH TYPŮ

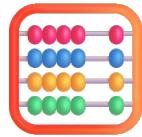
```
bool → int → float → complex
```

- `bool` je podtřída `int`
- `int` se povyšuje na `float`
- Python volí vždy „širší“ typ



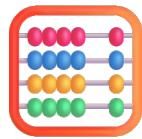
SMÍŠENÝ VÝPOČET

```
1 a = 3      # int
2 b = 4.5    # float
3
4 c = a * b
5
6 print(c)
7 print(type(c))
```



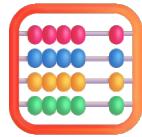
SMÍŠENÝ VÝPOČET

```
1 a = 3      # int
2 b = 4.5    # float
3
4 c = a * b
5
6 print(c)
7 print(type(c))
```



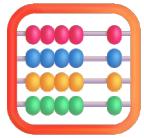
SMÍŠENÝ VÝPOČET

```
1 a = 3      # int
2 b = 4.5    # float
3
4 c = a * b
5
6 print(c)
7 print(type(c))
```



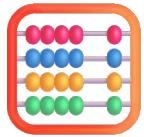
SMÍŠENÝ VÝPOČET

```
1 a = 3      # int
2 b = 4.5    # float
3
4 c = a * b
5
6 print(c)
7 print(type(c))
```



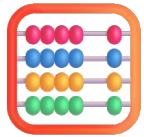
SMÍŠENÝ VÝPOČET

```
1 a = 3      # int
2 b = 4.5    # float
3
4 c = a * b
5
6 print(c)
7 print(type(c))
```



SMÍŠENÝ VÝPOČET

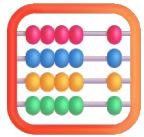
```
1 a = 3      # int
2 b = 4.5    # float
3
4 c = a * b
5
6 print(c)
7 print(type(c))
```



SMÍŠENÝ VÝPOČET

```
1 a = 3      # int
2 b = 4.5    # float
3
4 c = a * b
5
6 print(c)
7 print(type(c))
```

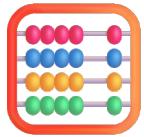
- a se dočasně převeze na 3.0



SMÍŠENÝ VÝPOČET

```
1 a = 3      # int
2 b = 4.5    # float
3
4 c = a * b
5
6 print(c)
7 print(type(c))
```

- a se dočasně převede na 3.0
- proběhne výpočet 3.0 * 4.5



SMÍŠENÝ VÝPOČET

```
1 a = 3      # int
2 b = 4.5    # float
3
4 c = a * b
5
6 print(c)
7 print(type(c))
```

- a se dočasně převeze na 3.0
- proběhne výpočet $3.0 * 4.5$
- výsledek je float



TYP SE NEMĚNÍ ZPĚTNĚ

```
1 a = 3
2 b = 4.5
3 c = a * b
4
5 print(type(a))
6 print(type(b))
7 print(type(c))
```



TYP SE NEMĚNÍ ZPĚTNĚ

```
1 a = 3
2 b = 4.5
3 c = a * b
4
5 print(type(a))
6 print(type(b))
7 print(type(c))
```



TYP SE NEMĚNÍ ZPĚTNĚ

```
1 a = 3
2 b = 4.5
3 c = a * b
4
5 print(type(a))
6 print(type(b))
7 print(type(c))
```



TYP SE NEMĚNÍ ZPĚTNĚ

```
1 a = 3
2 b = 4.5
3 c = a * b
4
5 print(type(a))
6 print(type(b))
7 print(type(c))
```



TYP SE NEMĚNÍ ZPĚTNĚ

```
1 a = 3
2 b = 4.5
3 c = a * b
4
5 print(type(a))
6 print(type(b))
7 print(type(c))
```



TYP SE NEMĚNÍ ZPĚTNĚ

```
1 a = 3
2 b = 4.5
3 c = a * b
4
5 print(type(a))
6 print(type(b))
7 print(type(c))
```



TYP SE NEMĚNÍ ZPĚTNĚ

```
1 a = 3
2 b = 4.5
3 c = a * b
4
5 print(type(a))
6 print(type(b))
7 print(type(c))
```



TYP SE NEMĚNÍ ZPĚTNĚ

```
1 a = 3
2 b = 4.5
3 c = a * b
4
5 print(type(a))
6 print(type(b))
7 print(type(c))
```

- a zůstává int



TYP SE NEMĚNÍ ZPĚTNĚ

```
1 a = 3
2 b = 4.5
3 c = a * b
4
5 print(type(a))
6 print(type(b))
7 print(type(c))
```

- a zůstává int
- cast probíhá jen uvnitř výrazu



bool JE VLASTNĚ int

```
1 print(True + 1)      # 2
2 print(True * 2.5)    # 2.5
3 print(isinstance(True, int)) # True
```



bool JE VLASTNĚ int

```
1 print(True + 1)      # 2
2 print(True * 2.5)    # 2.5
3 print(isinstance(True, int)) # True
```



bool JE VLASTNĚ int

```
1 print(True + 1)      # 2
2 print(True * 2.5)    # 2.5
3 print(isinstance(True, int)) # True
```



bool JE VLASTNĚ int

```
1 print(True + 1)      # 2
2 print(True * 2.5)    # 2.5
3 print(isinstance(True, int)) # True
```



bool JE VLASTNĚ int

```
1 print(True + 1)      # 2
2 print(True * 2.5)    # 2.5
3 print(isinstance(True, int)) # True
```

True → 1 → 1.0



SHRNUTÍ

Python při výpočtech automaticky povyšuje typy:

```
bool → int → float
```



SHRNUTÍ

Python při výpočtech automaticky povyšuje typy:

```
bool → int → float
```

Cílem je **neztratit přesnost**.



KOLIK MÍSTA ZABÍRAJÍ TYPY V PYTHONU?

Spoiler: mnohem víc než čekáš



DŮLEŽITÝ FAKT

V Pythonu je všechno objekt



DŮLEŽITÝ FAKT

- hodnota
- typ
- reference counter
- metadata



JAK TO ZMĚŘIT?

```
1 import sys  
2  
3 sys.getsizeof(obj)
```

Andrej Pčelovodov

BOOL

```
1 import sys  
2  
3 sys.getsizeof(True)
```

BOOL

28

Andrej Pčelovodov

INT

```
1 sys.getsizeof(0)
2 sys.getsizeof(123456)
```

INT

28

28

Andrej Pčelovodov

Arvion

VELKÁ ČÍSLA ROSTOU

```
1 sys.getsizeof(10**30)
```

Andrej Pčelovodov

VELKÁ ČÍSLA ROSTOU

32 (nebo víc)

Andrej Pčelovodov

FLOAT

```
1 sys.getsizeof(0.0)
```

Andrej Pčelovodov

FLOAT

24

Andrej Pčelovodov

STR

```
1 sys.getsizeof("")  
2 sys.getsizeof("a")  
3 sys.getsizeof("hello")
```

STR

49

50

54

Andrej Pčelovodov

Arvion



PŘEHLED (64-BIT CPYTHON)

typ	velikost
bool	~28 B
int	~28 B+
float	~24 B
empty str	~49 B



PRAKTICKÝ DOPAD

- 1 milion intů ≈ 28 MB
- Python není paměťově efektivní
- pro data \rightarrow NumPy / array / C++



PRAVIDLO Z PRAXE

malý počet hodnot → Python ok

miliony pixelů / čísel → použij
NumPy nebo C++



TEXTOVÝ TYP

str

```
1 str1 = 'Single quotes.'
2 str2 = "Double quotes."
3 str3 = '''Triple
4 quotes'''
5 str4 = """Triple
6 double-quotes."""
7
8 str(str4) # User friendly version of string
9 repr(str4) # Only for debugging
```



TEXTOVÝ TYP

str

```
1 str1 = 'Single quotes.'
2 str2 = "Double quotes."
3 str3 = '''Triple
4 quotes'''
5 str4 = """Triple
6 double-quotes."""
7
8 str(str4) # User friendly version of string
9 repr(str4) # Only for debugging
```



TEXTOVÝ TYP

str

```
1 str1 = 'Single quotes.'
2 str2 = "Double quotes."
3 str3 = '''Triple
4 quotes'''
5 str4 = """Triple
6 double-quotes."""
7
8 str(str4) # User friendly version of string
9 repr(str4) # Only for debugging
```



TEXTOVÝ TYP

str

```
1 str1 = 'Single quotes.'
2 str2 = "Double quotes."
3 str3 = '''Triple
4 quotes'''
5 str4 = """Triple
6 double-quotes."""
7
8 str(str4) # User friendly version of string
9 repr(str4) # Only for debugging
```



TEXTOVÝ TYP

str

```
1 str1 = 'Single quotes.'
2 str2 = "Double quotes."
3 str3 = '''Triple
4 quotes'''
5 str4 = """Triple
6 double-quotes."""
7
8 str(str4) # User friendly version of string
9 repr(str4) # Only for debugging
```



TEXTOVÝ TYP

str

```
1 str1 = 'Single quotes.'
2 str2 = "Double quotes."
3 str3 = '''Triple
4 quotes'''
5 str4 = """Triple
6 double-quotes."""
7
8 str(str4) # User friendly version of string
9 repr(str4) # Only for debugging
```



TEXTOVÝ TYP

str

```
1 str1 = 'Single quotes.'
2 str2 = "Double quotes."
3 str3 = '''Triple
4 quotes'''
5 str4 = """Triple
6 double-quotes."""
7
8 str(str4) # User friendly version of string
9 repr(str4) # Only for debugging
```

ZAJÍMAVOST O str

```
1 s = "Python"  
2 len(s)  
3 s.removeprefix("Py")  
4 s.removesuffix("on")  
5 print(s)  
6 s.removeprefix("User")
```

ZAJÍMAVOST O str

```
1 s = "Python"  
2 len(s)  
3 s.removeprefix("Py")  
4 s.removesuffix("on")  
5 print(s)  
6 s.removeprefix("User")
```

ZAJÍMAVOST O str

```
1 s = "Python"  
2 len(s)  
3 s.removeprefix("Py")  
4 s.removesuffix("on")  
5 print(s)  
6 s.removeprefix("User")
```

ZAJÍMAVOST O str

```
1 s = "Python"  
2 len(s)  
3 s.removeprefix("Py")  
4 s.removesuffix("on")  
5 print(s)  
6 s.removeprefix("User")
```

ZAJÍMAVOST O str

```
1 s = "Python"  
2 len(s)  
3 s.removeprefix("Py")  
4 s.removesuffix("on")  
5 print(s)  
6 s.removeprefix("User")
```

ZAJÍMAVOST O str

```
1 s = "Python"  
2 len(s)  
3 s.removeprefix("Py")  
4 s.removesuffix("on")  
5 print(s)  
6 s.removeprefix("User")
```

ZAJÍMAVOST O str

```
1 s = "Python"  
2 len(s)  
3 s.removeprefix("Py")  
4 s.removesuffix("on")  
5 print(s)  
6 s.removeprefix("User")
```

ZAJÍMAVOST O str

```
1 s = "Python"  
2 len(s)  
3 s.removeprefix("Py")  
4 s.removesuffix("on")  
5 print(s)  
6 s.removeprefix("User")
```



Stringy jsou **immutable**



ENCODING (KÓDOVÁNÍ TEXTU)

Encoding určuje, jak jsou znaky uloženy jako bajty v paměti nebo v souboru.



ENCODING (KÓDOVÁNÍ TEXTU)

Encoding určuje, jak jsou znaky uloženy jako bajty v paměti nebo v souboru.

- Počítač rozumí jen číslům (bajtům)

ENCODING (KÓDOVÁNÍ TEXTU)

Encoding určuje, jak jsou znaky uloženy jako bajty v paměti nebo v souboru.

- Počítač rozumí jen číslům (bajtům)
- Text je jen dohoda

ENCODING (KÓDOVÁNÍ TEXTU)

Encoding určuje, jak jsou znaky uloženy jako bajty v paměti nebo v souboru.

- Počítač rozumí jen číslům (bajtům)
- Text je jen dohoda
- Bez správného encodingu vzniká chaos



ASCII VS UTF-8

- ASCII – 128 znaků (angličtina)

A -> 65

č -> 11001000 10101101

😊 -> 11110000 10011111 10011000 10000000



ASCII VS UTF-8

- ASCII – 128 znaků (angličtina)
- UTF-8 – celý Unicode svět

A -> 65

č -> 11001000 10101101

😊 -> 11110000 10011111 10011000 10000000



ASCII VS UTF-8

- ASCII – 128 znaků (angličtina)
- UTF-8 – celý Unicode svět
- diakritika, emoji, asijské znaky

A → 65

č → 11001000 10101101

😊 → 11110000 10011111 10011000 10000000



UTF-8 V PYTHONU 3

V Pythonu 3 je **string Unicode**.

```
1 text = "Příliš žlutoučký kůň 🐾"
2 print(text)
3 print(type(text))
```



UTF-8 V PYTHONU 3

V Pythonu 3 je **string Unicode**.

```
1 text = "Příliš žlutoučký kůň 🐾"
2 print(text)
3 print(type(text))
```

- $\text{str} \neq \text{bajty}$



UTF-8 V PYTHONU 3

V Pythonu 3 je **string Unicode**.

```
1 text = "Příliš žlutoučký kůň 🐾"
2 print(text)
3 print(type(text))
```

- $\text{str} \neq \text{bajty}$
- žádný encoding uvnitř stringu



str VS bytes

```
1 s = "čau"
2 b = s.encode("utf-8")
3
4 print(s)          # text
5 print(b)          # bajty
```



str VS bytes

```
1 s = "čau"
2 b = s.encode("utf-8")
3
4 print(s)          # text
5 print(b)          # bajty
```

- str = znaky



str VS bytes

```
1 s = "čau"
2 b = s.encode("utf-8")
3
4 print(s)          # text
5 print(b)          # bajty
```

- str = znaky
- bytes = čísla 0–255



str VS bytes

```
1 s = "čau"
2 b = s.encode("utf-8")
3
4 print(s)          # text
5 print(b)          # bajty
```

- str = znaky
- bytes = čísla 0–255
- encode / decode je hranice



ENCODE VS DECODE

```
1 text = "žluťoučký"
2
3 data = text.encode("utf-8") # b'\xc5\xbelu\xc5\xa5ou\xc4\x8dk\xc3'
4 text2 = data.decode("utf-8") # žluťoučký
```



ENCODE VS DECODE

```
1 text = "žluťoučký"
2
3 data = text.encode("utf-8") # b'\xc5\xbelu\xc5\xa5ou\xc4\x8dk\xc3'
4 text2 = data.decode("utf-8") # žluťoučký
```



ENCODE VS DECODE

```
1 text = "žluťoučký"
2
3 data = text.encode("utf-8") # b'\xc5\xbelu\xc5\xa5ou\xc4\x8dk\xc3'
4 text2 = data.decode("utf-8") # žluťoučký
```



ENCODE VS DECODE

```
1 text = "žluťoučký"
2
3 data = text.encode("utf-8") # b'\xc5\xbelu\xc5\xa5ou\xc4\x8dk\xc3'
4 text2 = data.decode("utf-8") # žluťoučký
```



ENCODE VS DECODE

```
1 text = "žluťoučký"
2
3 data = text.encode("utf-8") # b'\xc5\xbelu\xc5\xa5ou\xc4\x8dk\xc3'
4 text2 = data.decode("utf-8") # žluťoučký
```

- encode: str → bytes



ENCODE VS DECODE

```
1 text = "žluťoučký"
2
3 data = text.encode("utf-8") # b'\xc5\xbelu\xc5\xa5ou\xc4\x8dk\xc3'
4 text2 = data.decode("utf-8") # žluťoučký
```

- encode: str → bytes
- decode: bytes → str



STRING JAKO SEKVENCE ZNAKŮ

V Pythonu je str **sekvence znaků**.



STRING JAKO SEKVENCE ZNAKŮ

V Pythonu je str **sekvence znaků**.

- má délku (`len()`)



STRING JAKO SEKVENCE ZNAKŮ

V Pythonu je str **sekvence znaků**.

- má délku (`len()`)
- lze indexovat



STRING JAKO SEKVENCE ZNAKŮ

V Pythonu je str **sekvence znaků**.

- má délku (`len()`)
- lze indexovat
- lze řezat (`slicing`)

1
2
3
4

INDEXOVÁNÍ STRINGU

```
1 text = "Python"
2
3 print(text[0])    # P
4 print(text[1])    # y
5 print(text[5])    # n
```

Andrej Pčelovodov

1
2
3
4

INDEXOVÁNÍ STRINGU

```
1 text = "Python"  
2  
3 print(text[0])    # P  
4 print(text[1])    # y  
5 print(text[5])    # n
```

Andrej Pčelovodov

1
2
3
4

INDEXOVÁNÍ STRINGU

```
1 text = "Python"
2
3 print(text[0])    # P
4 print(text[1])    # y
5 print(text[5])    # n
```

1
2
3
4

INDEXOVÁNÍ STRINGU

```
1 text = "Python"
2
3 print(text[0])    # P
4 print(text[1])    # y
5 print(text[5])    # n
```

1
2
3
4

INDEXOVÁNÍ STRINGU

```
1 text = "Python"
2
3 print(text[0])    # P
4 print(text[1])    # y
5 print(text[5])    # n
```

Andrej Pčelovodov

1
2
3
4

INDEXOVÁNÍ STRINGU

```
1 text = "Python"  
2  
3 print(text[0])    # P  
4 print(text[1])    # y  
5 print(text[5])    # n
```

- indexy začínají od 0

1
2
3
4

INDEXOVÁNÍ STRINGU

```
1 text = "Python"  
2  
3 print(text[0])    # P  
4 print(text[1])    # y  
5 print(text[5])    # n
```

- indexy začínají od 0
- poslední znak má index `len(text) - 1`



ZÁPORNÉ INDEXY

```
1 text = "Python"  
2  
3 print(text[-1]) # n  
4 print(text[-2]) # o
```



ZÁPORNÉ INDEXY

```
1 text = "Python"
2
3 print(text[-1]) # n
4 print(text[-2]) # o
```



ZÁPORNÉ INDEXY

```
1 text = "Python"
2
3 print(text[-1]) # n
4 print(text[-2]) # o
```



ZÁPORNÉ INDEXY

```
1 text = "Python"
2
3 print(text[-1]) # n
4 print(text[-2]) # o
```



ZÁPORNÉ INDEXY

```
1 text = "Python"  
2  
3 print(text[-1]) # n  
4 print(text[-2]) # o
```

- -1 = poslední znak



ZÁPORNÉ INDEXY

```
1 text = "Python"  
2  
3 print(text[-1]) # n  
4 print(text[-2]) # o
```

- -1 = poslední znak
- praktické a čitelné



STRING JE IMMUTABLE

```
1 text = "Python"  
2 text[0] = "J"
```



STRING JE IMMUTABLE

```
1 text = "Python"  
2 text[0] = "J"
```



STRING JE IMMUTABLE

```
1 text = "Python"  
2 text[0] = "J"
```



STRING JE IMMUTABLE

```
1 text = "Python"  
2 text[0] = "J"
```

 TypeError



STRING JE IMMUTABLE

```
1 text = "Python"  
2 text[0] = "J"
```

 TypeError

- string nelze měnit na místě



STRING JE IMMUTABLE

```
1 text = "Python"  
2 text[0] = "J"
```



TypeError

- string nelze měnit na místě
- vždy vzniká nový string



SLICING STRINGS

```
1 text = "Python"
2
3 print(text[0:4])    # Pyth
4 print(text[2:])     # on
5 print(text[:3])     # Pyt
```



SLICING STRINGS

```
1 text = "Python"
2
3 print(text[0:4])    # Pyth
4 print(text[2:])      # hon
5 print(text[:3])      # Pyt
```



SLICING STRINGS

```
1 text = "Python"
2
3 print(text[0:4])    # Pyth
4 print(text[2:])      # thon
5 print(text[:3])      # Pyt
```



SLICING STRINGS

```
1 text = "Python"
2
3 print(text[0:4])    # Pyth
4 print(text[2:])      # hon
5 print(text[:3])      # Pyt
```



SLICING STRINGS

```
1 text = "Python"
2
3 print(text[0:4])    # Pyth
4 print(text[2:])     # hon
5 print(text[:3])     # Pyt
```



SLICING STRINGU

```
1 text = "Python"  
2  
3 print(text[0:4])    # Pyth  
4 print(text[2:])     # thon  
5 print(text[:3])     # Pyt
```

- start je včetně



SLICING STRINGU

```
1 text = "Python"  
2  
3 print(text[0:4])    # Pyth  
4 print(text[2:])     # thon  
5 print(text[:3])     # Pyt
```

- start je včetně
- end je **exkluzivní**



SLICING + ZÁPORNÉ INDEXY

```
1 text = "Python"
2
3 print(text[:-1])    # Pytho
4 print(text[-3:])    # hon
```



SLICING + ZÁPORNÉ INDEXY

```
1 text = "Python"
2
3 print(text[:-1])    # Pytho
4 print(text[-3:])    # hon
```



SLICING + ZÁPORNÉ INDEXY

```
1 text = "Python"
2
3 print(text[:-1])    # Pytho
4 print(text[-3:])    # hon
```



SLICING + ZÁPORNÉ INDEXY

```
1 text = "Python"
2
3 print(text[:-1])    # Pytho
4 print(text[-3:])    # hon
```



SLICING + ZÁPORNÉ INDEXY

```
1 text = "Python"
2
3 print(text[:-1])    # Pytho
4 print(text[-3:])    # hon
```

- velmi čitelné



SLICING + ZÁPORNÉ INDEXY

```
1 text = "Python"
2
3 print(text[:-1])    # Pytho
4 print(text[-3:])    # hon
```

- velmi čitelné
- bez počítání délky



SLICING S KROKEM

```
1 text = "Python"
2
3 print(text[::-2])      # Pto
4 print(text[::-1])      # nohtyP
```

Andrej Pčelovodov



SLICING S KROKEM

```
1 text = "Python"
2
3 print(text[::-2])      # Pto
4 print(text[::-1])      # nohtyP
```



SLICING S KROKEM

```
1 text = "Python"
2
3 print(text[::-2])      # Pto
4 print(text[::-1])      # nohtyP
```



SLICING S KROKEM

```
1 text = "Python"
2
3 print(text[::-2])      # Pto
4 print(text[::-1])      # nohtyP
```



SLICING S KROKEM

```
1 text = "Python"  
2  
3 print(text[::-2])      # Pto  
4 print(text[::-1])      # nohtyP
```

- step = krok



SLICING S KROKEM

```
1 text = "Python"  
2  
3 print(text[::-2])      # Pto  
4 print(text[::-1])      # nohtyP
```

- step = krok
- záporný krok = obrácení



INDEXERROR

```
1 text = "Python"  
2 print(text[10]) # IndexError: string index out of range
```



INDEXERROR

```
1 text = "Python"  
2 print(text[10]) # IndexError: string index out of range
```



INDEXERROR

```
1 text = "Python"  
2 print(text[10]) # IndexError: string index out of range
```



INDEXERROR

```
1 text = "Python"  
2 print(text[10]) # IndexError: string index out of range
```

- index mimo rozsah



INDEXERROR

```
1 text = "Python"  
2 print(text[10]) # IndexError: string index out of range
```

- index mimo rozsah
- slicing je bezpečný



FORMÁTOVÁNÍ STRINGŮ V PYTHONU

Jak skládat text + proměnné čitelně a bezpečně

Andrej Pčelovodov

NAIVNÍ SPOJOVÁNÍ (NEDĚLEJ TOHLE)

```
1 name = "Andrej"  
2 age = 30  
3  
4 msg = "Jmenuji se " + name + " a je mi " + str(age) + " let."  
5 print(msg)
```



STARÝ STYL: % OPERÁTOR (C-LIKE)

```
1 name = "Andrej"  
2 age = 30  
3  
4 msg = "Jmenuji se %s a je mi %d let." % (name, age)  
5 print(msg)
```

Andrej Pčelovodov



STR.FORMAT()

```
1 name = "Andrej"  
2 age = 30  
3  
4 msg = "Jmenuji se {} a je mi {} let.".format(name, age)  
5 print(msg)
```



POJMENOVANÉ ARGUMENTY

```
1 msg = "Jmenuji se {name} a je mi {age} let.".format(  
2     name="Andrej",  
3     age=30  
4 )  
5 print(msg)
```



F-STRINGS (DOPORUČENÝ ZPŮSOB)

```
1 name = "Andrej"  
2 age = 30  
3  
4 msg = f"Jmenuji se {name} a je mi {age} let."  
5 print(msg)
```



VÝRAZY UVNITŘ F-STRINGU

```
1 a = 10
2 b = 5
3
4 print(f"Součet: {a + b}")
5 print(f"Dvojnásobek: {a * 2}")
6 print(f"Velkými: {'andrey'.upper()}")
```



VÝRAZY UVNITŘ F-STRINGU

```
1 a = 10
2 b = 5
3
4 print(f"Součet: {a + b}")
5 print(f"Dvojnásobek: {a * 2}")
6 print(f"Velkými: {'andrey'.upper()}")
```



VÝRAZY UVNITŘ F-STRINGU

```
1 a = 10
2 b = 5
3
4 print(f"Součet: {a + b}")
5 print(f"Dvojnásobek: {a * 2}")
6 print(f"Velkými: {'andrey'.upper()}")
```



VÝRAZY UVNITŘ F-STRINGU

```
1 a = 10
2 b = 5
3
4 print(f"Součet: {a + b}")
5 print(f"Dvojnásobek: {a * 2}")
6 print(f"Velkými: {'andrey'.upper()}")
```



VÝRAZY UVNITŘ F-STRINGU

```
1 a = 10
2 b = 5
3
4 print(f"Součet: {a + b}")
5 print(f"Dvojnásobek: {a * 2}")
6 print(f"Velkými: {'andrey'.upper()}")
```



VÝRAZY UVNITŘ F-STRINGU

```
1 a = 10
2 b = 5
3
4 print(f"Součet: {a + b}")
5 print(f"Dvojnásobek: {a * 2}")
6 print(f"Velkými: {'andrey'.upper()}")
```



FORMÁTOVÁNÍ ČÍSEL

```
1 price = 1234.56789
2
3 print(f"{price:.2f}")      # 2 desetinná místa
4 print(f"{price:10.2f}")    # zarovnání
5 print(f"{price:,}")        # oddělovače tisíců
```



FORMÁTOVÁNÍ ČÍSEL

```
1 price = 1234.56789
2
3 print(f"{price:.2f}")      # 2 desetinná místa
4 print(f"{price:10.2f}")    # zarovnání
5 print(f"{price:,}")        # oddělovače tisíců
```



FORMÁTOVÁNÍ ČÍSEL

```
1 price = 1234.56789
2
3 print(f"{price:.2f}")      # 2 desetinná místa
4 print(f"{price:10.2f}")    # zarovnání
5 print(f"{price:,}")        # oddělovače tisíců
```



FORMÁTOVÁNÍ ČÍSEL

```
1 price = 1234.56789
2
3 print(f"{price:.2f}")      # 2 desetinná místa
4 print(f"{price:10.2f}")    # zarovnání
5 print(f"{price:,}")        # oddělovače tisíců
```



FORMÁTOVÁNÍ ČÍSEL

```
1 price = 1234.56789
2
3 print(f"{price:.2f}")      # 2 desetinná místa
4 print(f"{price:10.2f}")    # zarovnání
5 print(f"{price:,}")        # oddělovače tisíců
```



DEBUG F-STRING (PYTHON 3.8+)

```
1 x = 42
2 y = 7
3
4 print(f"{x=}, {y=}, {x*y=}")
```



DEBUG F-STRING (PYTHON 3.8+)

```
1 x = 42
2 y = 7
3
4 print(f"{x=}, {y=}, {x*y=}")
```



DEBUG F-STRING (PYTHON 3.8+)

```
1 x = 42
2 y = 7
3
4 print(f"{x=}, {y=}, {x*y=}")
```



DEBUG F-STRING (PYTHON 3.8+)

```
1 x = 42
2 y = 7
3
4 print(f"{x=}, {y=}, {x*y=}")
```



DOPORUČENÍ Z PRAXE

- Nový kód → vždy f-strings
- Starý projekt → můžeš narazit na format()
- % styl už nepoužívat



KOLEKCE

LIST	TUPLE	DICTIONARY	SET
Allows duplicate members	Allows duplicate members	No duplicate members	No duplicate members
Changeble	Not changeable	Changeable, indexed	Cannot be changed, but can be added, non -indexed
Ordered	Ordered	Unordered	Unordered
Square bracket []	Round brackets ()	Curly brackets{ }	Curly brackets{ }

- list
- tuple



LIST (SEZNAM)

Nejzákladnější kolekce v Pythonu

Python List Methods

Input	Method	Output
[10, 20, 30]	.append(40)	[10, 20, 30, 40]
[10, 20, 30]	.extend([40])	[10, 20, 30, 40]
[10, 20, 30]	.insert(2, 40)	[10, 20, 40, 30]
[10, 20, 30, 20]	.count(20)	2
[10, 20, 30]	.clear()	[]
[10, 20, 30, 40]	.index(40)	3
[10, 20, 30, 40]	.remove(10)	[20, 30, 40]
[10, 20, 30, 40]	.pop(2)	[10, 20, 40]
[30, 20, 10]	.reverse()	[10, 20, 30]
[30, 10, 40, 20]	.sort()	[10, 20, 30, 40]
[10, 20, 30]	.copy()	[10, 20, 30]



LIST (SEZNAM)

```
1 numbers = [1, 2, 3, 4]
2 names = ["Anna", "Bob"]
3 mixed = [1, "text", True]
```



CO JE LIST UVNITŘ?

Dynamické pole (dynamic array)



CO JE LIST UVNITŘ?

```
[index]  
[0][1][2][3][4]  
↑ ↑ ↑  
paměť vedle sebe
```



KDY POUŽÍVAT LIST

- pořadí prvků je důležité
- často indexuješ
- procházíš for-cyklem
- přidáváš na konec



TYPICKÉ OPERACE (RYCHLÉ)

```
1 lst.append(10)      # O(1)
2 lst.pop()           # O(1)
3 lst[i]              # O(1)
4 len(lst)            # O(1)
```



CO JE POMALÉ

```
1 lst.insert(0, 10)
2 lst.pop(0)
3 lst.remove(x)
```

Andrej Pčelovodov



CO JE POMALÉ

všechny prvky se musí posunout $\rightarrow O(n)$

KDY LIST NEPOUŽÍVAT

- rychlé vyhledávání hodnot → použij set
- mapování klíč → hodnota → dict
- FIFO fronta → deque
- neměnná data → tuple



ČASTÁ CHYBA #1

```
1 if x in big_list:  
2     ...
```



ČASTÁ CHYBA #1

membership test je $O(n)$

```
big_set = set(big_list)  # O(1)
```



PYTHONIC STYL

```
1 squares = [x*x for x in range(10)]  
2 evens   = [x for x in data if x % 2 == 0]
```



SHRNUTÍ

- list = dynamické pole
- rychlý přístup přes index
- rychlé operace na konci
- pomalé operace uprostřed/na začátku
- není univerzální - vybírej správnou strukturu



PRAKTICKÉ PRAVIDLO

Pokud jen ukládáš věci za sebe → list.
Jakmile řešíš výkon nebo vyhledávání → zamysli se.



TUPLE

Neměnná (immutable) sekvence hodnot

methods	example	description
a.index(tuple)	>>> a=(1,2,3,4,5) >>> a.index(5) 4	Returns the index of the first matched item.
a.count(tuple)	>>>a=(1,2,3,4,5) >>> a.count(3) 1	Returns the count of the given element.
len(tuple)	>>> len(a) 5	return the length of the tuple
min(tuple)	>>> min(a) 1	return the minimum element in a tuple
max(tuple)	>>> max(a) 5	return the maximum element in a tuple
del(tuple)	>>> del(a)	Delete the entire tuple.



TUPLE

```
1 t = (1, 2, 3)
2 names = ("Anna", "Bob")
3 point = (10, 20)
```



ROZDÍL OPROTI LISTU?

list → **mutable**

tuple → **immutable**



CO ZNAMENÁ IMMUTABLE?

```
1 t = (1, 2, 3)
2 t[0] = 10
```

Andrej Pčelovodov



CO ZNAMENÁ IMMUTABLE?

TypeError:

'tuple' object does not support item assignment

Andrej Pčelovodov



CO TUPLE UMÍ

- indexování
- slicing
- iterace
- len()



CO TUPLE UMÍ

```
1 t = (10, 20, 30)
2
3 t[0]
4 t[1:3]
5 for x in t:
6     print(x)
```



CO TUPLE NEUMÍ

```
t.append(4)  
t.remove(2)  
t[0] = 10
```



PROČ TUPLE VŮBEC EXISTUJE?

Když se data nemají měnit,
řekni to typem.



KDY TUPLE POUŽÍVAT

- souřadnice (x, y)
- RGB barva
- konfigurace
- návrat více hodnot z funkce
- konstantní data



TYPICKÝ PŘÍKLAD

```
1 point = (10, 20)
2
3 x, y = point
4 print(x, y)
```



VÍCE NÁVRATOVÝCH HODNOT

```
1 def divide(a, b):  
2     return a // b, a % b  
3  
4 q, r = divide(10, 3)
```



VÝHODY TUPLE

- bezpečnější (žádné side effects)
- rychlejší než list
- menší paměť
- může být klíčem v dict/set



TUPLE JAKO KLÍČ

```
1 grid = {}  
2  
3 grid[(10, 20)] = "tree"  
4 grid[(5, 8)] = "house"
```



POZOR NA JEDNU VĚC

```
1 t = (1, [2, 3])
2 t[1].append(4)
3 print(t)
```



POZOR NA JEDNU VĚC

([1](#), [[2](#), [3](#), [4](#)])

👉 tuple je immutable, ale objekty uvnitř nemusí být



SHRNUTÍ

- tuple = neměnný list
- bezpečnější a předvídatelnější
- lepší pro konstantní data
- hashovatelný → může být klíč
- když se to nemá měnit, nepoužívej list



PRAKTICKÉ PRAVIDLO

Pokud plánuješ měnit → list.

Pokud ne → tuple.

Jednoduché. Čitelné. Bez překvapení.



SET (MNOŽINA)

Kolekce unikátních hodnot bez pořadí

Python Set Methods

Python has a set of built-in methods that you can use on sets.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two or more sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with another set, or any other iterable



SET (MNOŽINA)

```
1 s = {1, 2, 3}  
2 names = {"Anna", "Bob", "Eva"}
```



AUTOMATICKY ODSTRAŇUJE DUPLICITY

```
1 s = {1, 1, 2, 2, 3}  
2 print(s)
```



AUTOMATICKY ODSTRAŇUJE DUPLICITY

```
{1, 2, 3}
```



CO JE SET UVNITŘ?

Hash tabulka (hash table)



CO TO ZNAMENÁ PRAKTICKY?

vyhledávání → $O(1)$

přidání → $O(1)$

mazání → $O(1)$



TYPICKÉ POUŽITÍ

```
1 allowed = {"admin", "manager"}  
2  
3 if user_role in allowed:  
4     allow_access()
```



ŠPATNĚ (POMALÉ)

```
1 allowed = ["admin", "manager"]  
2  
3 if user_role in allowed: # O(n)  
4 ...
```



ŠPATNĚ (POMALÉ)

list musí projít všechny prvky



ZÁKLADNÍ OPERACE

```
1 s.add(10)
2 s.remove(2)
3 s.discard(5)
4 len(s)
```



SÍLA SETU = MNOŽINOVÁ MATEMATIKA

operace jako ve škole

Andrej Pčelovodov



MNOŽINOVÉ OPERACE

```
1 a = {1, 2, 3}
2 b = {3, 4, 5}
3
4 a | b    # union
5 a & b    # intersection
6 a - b    # difference
```



VÝSLEDEK

```
a | b → {1,2,3,4,5}  
a & b → {3}  
a - b → {1,2}
```



DŮLEŽITÉ OMEZENÍ

prvky musí být hashovatelné
(immutable)



DŮLEŽITÉ OMEZENÍ

```
1 {1, 2, 3}          # OK
2 {"a", "b"}         # OK
3 {[1, 2]}          # ✗
```



PROČ?

mutable objekty mění hash → rozbije to hash tabulku



TYPICKÝ TRIK: ODSTRANĚNÍ DUPLICIT

```
1 data = [1, 2, 2, 3, 3, 3]
2
3 unique = list(set(data))
```



SHRNUTÍ

- set = unikátní hodnoty
- žádné pořadí
- extrémně rychlé vyhledávání
- ideální pro membership testy
- nefunguje s mutable objekty



PRAKTICKÉ PRAVIDLO

Potřebuješ pořadí → list.

Potřebuješ rychle hledat → set.

Nepoužívej list jako set. Nikdy.



DICT (SLOVNÍK)

mapování
klíč → hodnota

Python Dictionary Methods

- dict •
 - clear()
 - copy()
 - fromkeys()
 - get()
 - items()
 - keys()
 - pop()
 - popitem()
 - setdefault()
 - update()
 - values()

**Python
Dictionary
Methods**





DICT (SLOVNÍK)

```
1 user = {  
2     "name": "Anna",  
3     "age": 25,  
4     "admin": True  
5 }
```



CO JE DICT UVNITŘ?

Hash tabulka (hash table)



CO TO ZNAMENÁ?

lookup	→ $O(1)$
insert	→ $O(1)$
delete	→ $O(1)$

Andrej Pčelovodov



ZÁKLADNÍ OPERACE

```
1 user["name"]  
2 user["age"] = 26  
3 del user["admin"]  
4 len(user)
```



PROČ JE DICT TAK DŮLEŽITÝ?

přístup podle klíče je mnohem rychlejší
než hledání v listu

ŠPATNĚ (POMALÉ)

```
1 users = [
2     ("anna", 25),
3     ("bob", 30)
4 ]
5
6 for name, age in users:
7     if name == "bob":
8         print(age)
```



SPRÁVNĚ

```
1 users = {  
2     "anna": 25,  
3     "bob": 30  
4 }  
5  
6 print(users["bob"])
```



POŘADÍ V DICTU

Python 3.7+ zachovává pořadí vložení

Andrej Pčelovodov

NEW

POŘADÍ V DICTU

```
1 d = {}
2
3 d["a"] = 1
4 d["b"] = 2
5 d["c"] = 3
6
7 print(d)
```

Andrej Pčelovodov



ITERACE

```
1 for key in d:  
2     ...  
3  
4 for value in d.values():  
5     ...  
6  
7 for k, v in d.items():  
8     ...
```



POZOR NA KEYERROR

```
1 age = users["petr"]
```

Andrej Pčelovodov



BEZPEČNĚJI

```
1 age = users.get("petr", 0)
```

Andrej Pčelovodov



UŽITEČNÉ METODY

- 1 d.get(key)
- 2 d.setdefault(key, value)
- 3 d.update(other)
- 4 d.pop(key)



OMEZENÍ

klíče musí být hashovatelné
(immutable)



CO FUNGUJE / NEFUNGUJE

```
1 {"a": 1}          # OK
2 {(1,2): 3}      # OK
3 {[1,2]: 3}     # ✗
```



TYPICKÉ POUŽITÍ V PRAXI

- JSON data
- konfigurace
- počítání výskytů
- cache
- indexování objektů podle ID



POČÍTÁNÍ VÝSKYTŮ (KLASIKA)

```
1 counts = {}
2
3 for word in words:
4     counts[word] = counts.get(word, 0) + 1
```



SHRNUTÍ

- dict = klíč → hodnota
- velmi rychlé vyhledávání
- zachovává pořadí
- nejpoužívanější struktura v Pythonu
- klíče musí být immutable



PRAKTICKÉ PRAVIDLO

Když hledáš podle jména, ID nebo klíče → dict.

Pokud používáš list a ručně hledáš hodnotu, pravděpodobně používáš špatnou strukturu.

ENUM (ENUMERATION)

pojmenované konstanty místo „magických čísel“

```
•  
from enum import Enum  
  
class Direction(Enum):  
    UP = 1  
    DOWN = 2  
    LEFT = 3  
    RIGHT = 4
```



PROBLÉM BEZ ENUMU

```
1 camera_type = 2
2 event = 7
3
4 if camera_type == 2:
5     ...
```

PROBLÉM BEZ ENUMU

? 2 znamená co?

? 7 znamená co?

nečitelné, křehké, plné bugů



ŘEŠENÍ: ENUM

```
1 from enum import Enum
```



DEFINICE ENUMU

```
1 from enum import Enum  
2  
3 class CameraType(Enum):  
4     CCD = 1  
5     CMOS = 2  
6     SIMULATOR = 3
```



POUŽITÍ

```
1 camera = CameraType.CMOS
2
3 if camera == CameraType.CMOS:
4     print("CMOS camera")
```



CO JE ENUM UVNITŘ?

normální třída + singleton objekty



ENUM NENÍ JEN ČÍSLO

```
1 print(CameraType.CMOS)
2 print(CameraType.CMOS.name)
3 print(CameraType.CMOS.value)
```



VÝSTUP

CameraType . CMOS

CMOS

2

Andrej Pčelovodov



PŘÍKLAD: TYPY KAMER

reálný use-case z praxe



CAMERATYPE

```
1 class CameraType(Enum):  
2     CCD = 1  
3     CMOS = 2  
4     USB = 3  
5     NETWORK = 4
```



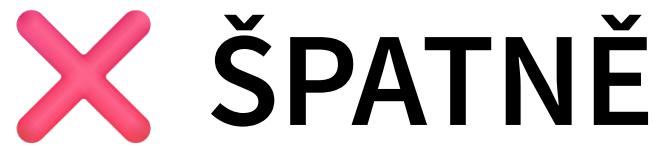
POUŽITÍ V APLIKACI

```
1 def init_camera(cam_type: CameraType):  
2     if cam_type == CameraType.CCD:  
3         init_ccd()  
4     elif cam_type == CameraType.CMOS:  
5         init_cmos()
```



PŘÍKLAD: GUI EVENTY

místo stringů nebo čísel



```
1 if event == "click":  
2 if event == "CLICK":  
3 if event == "clik":
```



EVENT ENUM

```
1 class EventType(Enum):  
2     CLICK = 1  
3     DOUBLE_CLICK = 2  
4     KEY_PRESS = 3  
5     WINDOW_CLOSE = 4
```



POUŽITÍ

```
1 def handle_event(event: EventType):  
2     if event == EventType.CLICK:  
3         ...  
4     elif event == EventType.KEY_PRESS:  
5         ...
```



AUTO HODNOTY

```
1 from enum import auto
2
3 class State(Enum):
4     IDLE = auto()
5     RUNNING = auto()
6     STOPPED = auto()
```



```
1 from enum import IntEnum
2
3 class ErrorCode(IntEnum):
4     OK = 0
5     FAIL = 1
6     TIMEOUT = 2
```



SHRNUTÍ

- žádná magická čísla
- lepší čitelnost
- méně bugů
- lepší autocomplete
- ideální pro stavy, typy, eventy



PRAKTICKÉ PRAVIDLO

Pokud máš pevný seznam možností
→ použij Enum.



JAKÉ DATOVÉ TYPY KDE POUŽÍT?

Python ≠ C++

Andrej Pčelovodov

REALITA

Python objekty jsou velké a pomalé.
Hardware + data = potřebuješ efektivitu.

ZLATÉ PRAVIDLO

logika → Python

data → NumPy / array / C++

Andrej Pčelovodov



KAMERY – ŘÍZENÍ ZAŘÍZENÍ

malé množství hodnot, stavů,
konfigurace



POUŽÍVEJ

- 1 `Enum` # typ kamery, mód
- 2 `bool` # zap/vyp
- 3 `int` # expoziční čas, gain
- 4 `dict` # config
- 5 `dataclass` # struktury nastavení



NEPOUŽÍVEJ

```
list milionů hodnot  
string "magic constants"
```

Andrej Pčelovodov



IMAGE PROCESSING

miliony pixelů = miliony čísel



ŠPATNĚ (ČISTÝ PYTHON)

```
1 pixels = [[0]*1920 for _ in range(1080)]
```

PROČ JE TO ŠPATNĚ?

- každý pixel = Python int (~28 B)
- obrovská RAM
- pomalé cache
- žádný SIMD (Single Instruction, Multiple Data)



SPRÁVNĚ

```
1 import numpy as np  
2  
3 img = np.zeros((1080, 1920), dtype=np.uint8)
```

Andrej Pčelovodov

VÝHODY NUMPY

Andrej Pčelovodov

VÝHODY NUMPY

-  1 pixel = 1 byte (záleží na datovém typu)

VÝHODY NUMPY

-  1 pixel = 1 byte (záleží na datovém typu)
-  kontinuální paměť

VÝHODY NUMPY

-  1 pixel = 1 byte (záleží na datovém typu)
-  kontinuální paměť
-  C rychlosť

VÝHODY NUMPY

-  1 pixel = 1 byte (záleží na datovém typu)
-  kontinuální paměť
-  C rychlosť
-  OpenCV kompatibilita



STAGE / MOTORY / HARDWARE

stavy + fronty příkazů + telemetrie



POUŽÍVEJ

```
1 Enum          # stav motoru
2 float         # pozice
3 tuple          # (x, y, z)
4 deque          # fronta příkazů (double-ended queue (oboustranná fronta))
5 dataclass      # struktura stavu
```

PŘÍKLAD

```
1 class StageState(Enum):  
2     IDLE = 0  
3     MOVING = 1  
4     ERROR = 2  
5  
6 position: tuple[float, float, float]
```



RYCHLÁ MAPA ROZHODOVÁNÍ

situace

typ

stavy / módy

Enum

konfigurace

dict / dataclass

malé kolekce

list / tuple

unikátní hodnoty

set

pixely / signály

NumPy array

max výkon

C++ / C extension



PRAKTICKÁ FILOZOFIE

Python je dirigent orchestru.
Těžkou práci atď dělá C/NumPy.



DOMÁCÍ ÚKOL

Andrej Pčelovodov



KLÁVESOVÉ ZKRATKY = RYCHLOST

Myš je pomalá.

Klávesnice je násobně rychlejší.



KLÁVESOVÉ ZKRATKY = RYCHLOST

Myš je pomalá.

Klávesnice je násobně rychlejší.

Profesionálové téměř nesahají na myš.



NOVÁ VIRTUÁLNÍ PLOCHA

Win + Tab

Andrej Pčelovodov



NOVÁ VIRTUÁLNÍ PLOCHA

Win + Tab

→ vytvoříš nový desktop



NOVÁ VIRTUÁLNÍ PLOCHA

Win + Tab

→ vytvoříš nový desktop

→ oddělíš práci / školu / osobní věci



PŘEPÍNÁNÍ MEZI PLOCHAMI

Win + Ctrl + < / >



PŘEPÍNÁNÍ MEZI PLOCHAMI

Win + Ctrl + < / >

Okamžité přepnutí bez myši



PŘEPÍNÁNÍ MEZI PLOCHAMI

Win + Ctrl + < / >

Okamžité přepnutí bez myši

funguje i při fullscreen aplikacích



POHYB PO SLOVECH

Ctrl + < / >



POHYB PO SLOVECH

Ctrl + < / >

Skok po celých slovech



POHYB PO SLOVECH

Ctrl + < / >

Skok po celých slovech

Ne po jednom znaku (to je pro amatéry)



VÝBĚR TEXTU PO SLOVECH

Ctrl + Shift + < / >



VÝBĚR TEXTU PO SLOVECH

Ctrl + Shift + < / >

Rychlé označení celých slov



VÝBĚR TEXTU PO SLOVECH

Ctrl + Shift + < / >

Rychlé označení celých slov

ideální pro refactoring



MAZÁNÍ CELÝCH SLOV

Ctrl + Backspace



MAZÁNÍ CELÝCH SLOV

Ctrl + Backspace

Smaže celé slovo najednou



MAZÁNÍ CELÝCH SLOV

Ctrl + Backspace

Smaže celé slovo najednou

ne jen jeden znak