OMM

Andrej Pčelovodov

Arvion

OMM

if 💠 match

Andrej Pčelovodov

Arvion

OMM

if 💠 match

for 💠 while

Andrej Pčelovodov

Arvion

OMM

if 🔷 match

for 🔷 while

itertools

Andrej Pčelovodov

Arvion

FUNCTIONS

OD ZÁKLADŮ AŽ PO POKROČILÉ POUŽITÍ

Andrej Pčelovodov

Arvion

PROČ VŮBEC FUNKCE?

- Opakovatelnost kódu
- Čitelnost
- Oddělení odpovědnosti
- Testovatelnost

DEFINICE FUNKCE

```
1 def function_name(parameters):
2     """Optional docstring""" # Accessible via .__doc__
3     # Function body
4     return [expression]
```

DEFINICE FUNKCE

```
1 def function_name(parameters):
2     """Optional docstring""" # Accessible via .__doc__
3     # Function body
4     return [expression]
```

Andrej Pčelovodov

Arvion

DEFINICE FUNKCE

```
1 def function_name(parameters):
2     """Optional docstring""" # Accessible via .__doc__
3     # Function body
4     return [expression]
```

DEFINICE FUNKCE

```
1 def function_name(parameters):
2     """Optional docstring""" # Accessible via .__doc__
3     # Function body
4     return [expression]
```

DEFINICE FUNKCE

```
1 def function_name(parameters):
2     """Optional docstring""" # Accessible via .__doc__
3     # Function body
4     return [expression]
```

DEFINICE FUNKCE

DEFINICE FUNKCE

- **def**: Toto klíčové slovo zahajuje definici funkce.

DEFINICE FUNKCE

- **def**: Toto klíčové slovo zahajuje definici funkce.
- **function_name**: Název, který zvolíte pro funkci a který by měl popisovat její účel.

DEFINICE FUNKCE

- **def**: Toto klíčové slovo zahajuje definici funkce.
- **function_name**: Název, který zvolíte pro funkci a který by měl popisovat její účel.
- **parameters**: Volitelné vstupy, které funkce může přijmout, umístěné v závorkách.

DEFINICE FUNKCE

- **def**: Toto klíčové slovo zahajuje definici funkce.
- **function_name**: Název, který zvolíte pro funkci a který by měl popisovat její účel.
- **parameters**: Volitelné vstupy, které funkce může přijmout, umístěné v závorkách.
- **dvojtečka (:)**: Označuje začátek těla funkce.

DEFINICE FUNKCE

- **def**: Toto klíčové slovo zahajuje definici funkce.
- **function_name**: Název, který zvolíte pro funkci a který by měl popisovat její účel.
- **parameters**: Volitelné vstupy, které funkce může přijmout, umístěné v závorkách.
- **dvojtečka (:)**: Označuje začátek těla funkce.
- **docstring ("""Description""")**: Řetězec, který popisuje, co funkce dělá, je volitelný, ale pro přehlednost se doporučuje.

DEFINICE FUNKCE

- **def**: Toto klíčové slovo zahajuje definici funkce.
- **function_name**: Název, který zvolíte pro funkci a který by měl popisovat její účel.
- **parameters**: Volitelné vstupy, které funkce může přijmout, umístěné v závorkách.
- **dvojtečka (:)**: Označuje začátek těla funkce.
- **docstring ("""Description""")**: Řetězec, který popisuje, co funkce dělá, je volitelný, ale pro přehlednost se doporučuje.
- **return statement**: Používá se k odeslání hodnoty zpět volajícímu. Pokud je vynechán, funkce vrátí **None**.

VOLÁNÍ FUNKCE

```
1 def greet():
2     print("Hello world")
3
4 greet()
```

Volání funkce bez parametrů

PARAMETRY

```
1 def greet(name):
2     print(f"Hello {name}")
3
4 greet("Andrej")
```

RETURN HODNOTY

```
1 def add(a, b):
2     return a + b
3
4 result = add(3, 4)
5 print(result)
```

IMPLICITNÍ NÁVRAT NONE

```
1 def do_something():
2     x = 5
3     # Tady funkce vráti defaultně None
4
5 print(do_something())
```

EXPLICITNÍ NÁVRAT NONE

```
1 def do_something():
2     x = 5
3     return
4
5 print(do_something())
```

VÝCHOZÍ (DEFAULT) PARAMETRY

```
1 def power(base, exponent=2):
2     return base ** exponent
3
4 print(power(3))
5 print(power(3, 3))
```

VÝCHOZÍ (DEFAULT) PARAMETRY

```
1 def power(base, exponent=2):
2     return base ** exponent
3
4 print(power(3))
5 print(power(3, 3))
```

Andrej Pčelovodov

Arvion

VÝCHOZÍ (DEFAULT) PARAMETRY

```
1 def power(base, exponent=2):
2     return base ** exponent
3
4 print(power(3))
5 print(power(3, 3))
```

Andrej Pčelovodov

Arvion

VÝCHOZÍ (DEFAULT) PARAMETRY

```
1 def power(base, exponent=2):
2     return base ** exponent
3
4 print(power(3))
5 print(power(3, 3))
```

POZOR NA MUTABLE DEFAULT ARGUMENT

```
1 def add_item(item, items=[]):
2     items.append(item)
3     return items
4
5 print(add_item("A")) # ['A']
6 print(add_item("B")) # ['A', 'B']
```

POZOR NA MUTABLE DEFAULT ARGUMENT

```
1 def add_item(item, items=[]):
2     items.append(item)
3     return items
4
5 print(add_item("A")) # ['A']
6 print(add_item("B")) # ['A', 'B']
```

Andrej Pčelovodov

POZOR NA MUTABLE DEFAULT ARGUMENT

```python
def add_item(item, items=[]):
    items.append(item)
    return items

print(add_item("A")) # ['A']
print(add_item("B")) # ['A', 'B']
```

Seznam je sdílený mezi voláními.
Správně by mělo být **items=None**
a vytvořit nový seznam uvnitř.

*ARGS A **KWARGS

```
1 def print_all(*args):
2     for arg in args:
3         print(arg)
4
5 print_all(1, 2, 3)
```

*ARGS A **KWARGS

```python
def print_all(*args):
    for arg in args:
        print(arg)

print_all(1, 2, 3)
```

*args = tuple

**kwargs = dictionary

Používá se u API, frameworků, flexibilních funkcí.

KEYWORD ARGUMENTY

```
1 def introduce(name, age):
2     print(f"{name} is {age} years old")
3
4 introduce(age=25, name="Eva")
```

KEYWORD ARGUMENTY

```
1 def introduce(name, age):
2     print(f"{name} is {age} years old")
3
4 introduce(age=25, name="Eva")
```

Python umožňuje měnit pořadí pomocí pojmenovaných argumentů.

FUNKCE JSOU OBJEKTY

```
1 def greet(name):
2     return f"Hello {name}"
3
4 say_hello = greet
5
6 print(say_hello("World"))
```

FUNKCE JSOU OBJEKTY

```python
def greet(name):
    return f"Hello {name}"

say_hello = greet

print(say_hello("World"))
```

Funkce jsou **first-class objects**
(použití objektu není nijak omezeno).
Můžeme je přiřazovat, předávat, vracet.

FUNKCE JAKO ARGUMENT

```
1 def operate(func, x, y):
2     return func(x, y)
3
4 def multiply(a, b):
5     return a * b
6
7 print(operate(multiply, 3, 4))
```

LAMBDA FUNKCE

```
1 square = lambda x: x * x
2
3 print(square(5))
```

LAMBDA FUNKCE

```
1 square = lambda x: x * x
2
3 print(square(5))
```

Lambda je **anonymní funkce**.

Používat střídmě – čitelnost je důležitější než zkratky.

REKURZE 🚫

```
1 def factorial(n):
2     if n == 1:
3         return 1
4     return n * factorial(n - 1)
5
6 print(factorial(5))
```

REKURZE 🚫

```python
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n - 1)

print(factorial(5))
```

Rekurze musí mít ukončovací podmínku!

Python má omezený recursion depth.

Hrozí nebezpečí stack overflow.

Lepší je to vidět jednou a hned zapomenout.

SCOPE (LEGB PRAVIDLO)

```
1 x = 10
2
3 def test():
4     x = 5
5     print(x)
6
7 test()
8 print(x)
```

Local, Enclosing, Global, Built-in.

NESTED FUNKCE

```
1 def outer():
2     def inner():
3         print("Inside inner")
4     inner()
5
6 outer()
```

CLOSURE

```
1 def make_multiplier(factor):
2     def multiply(x):
3         return x * factor
4     return multiply
5
6 double = make_multiplier(2)
7 print(double(5))
```

DECORATORS

```
1  def logger(func):
2      def wrapper(*args, **kwargs):
3          print("Calling function...")
4          return func(*args, **kwargs)
5      return wrapper
6
7  @logger
8  def say_hi():
9      print("Hi")
10
11 say_hi()
```

TYPE HINTS

```
1 def add(a: int, b: int) -> int:
2     return a + b
```

DOCSTRING

```python
def add(a: int, b: int) -> int:
    """
    Returns sum of two integers.
    """
    return a + b
```

REÁLNÝ PŘÍKLAD – VALIDACE UŽIVATELE

```python
def is_valid_user(username: str) -> bool:
    if len(username) < 3:
        return False
    if not username.isalnum():
        return False
    return True

print(is_valid_user("andrej123"))
```

SHRNUTÍ

- Funkce jsou základ modularity
- Jsou objekty
- Podporují rekurzi, closures, dekorátory
- Bez nich není seriózní software