

CONDITIONALS & ITERATION

ŘÍZENÍ TOKU PROGRAMU V PYTHONU

CONDITIONALS & ITERATION

ŘÍZENÍ TOKU PROGRAMU V PYTHONU

if • match • for • while • itertools

PROČ JE TO DŮLEŽITÉ?

PROČ JE TO DŮLEŽITÉ?

- Rozhodování (podmínky)

PROČ JE TO DŮLEŽITÉ?

- Rozhodování (podmínky)
- Opakování práce (cykly)

PROČ JE TO DŮLEŽITÉ?

- Rozhodování (podmínky)
- Opakování práce (cykly)
- Čistý a čitelný kód

IF / ELIF / ELSE

```
1 age = 18
2
3 if age >= 18:
4     print("Dospělý")
5 elif age >= 15:
6     print("Téměř")
7 else:
8     print("Dítě")
```

VNOŘENÉ PODMÍNKY (NESTED IF)

```
1 if user_logged:  
2     if is_admin:  
3         print("Admin panel")
```


VNOŘENÉ PODMÍNKY (NESTED IF)

```
1 if user_logged:  
2     if is_admin:  
3         print("Admin panel")
```

👉 raději zjednodušovat a používat guard clauses

VNOŘENÉ PODMÍNKY (NESTED IF)

```
1 if user_logged:
2     if is_admin:
3         print("Admin panel")
```

👉 raději zjednodušovat a používat guard clauses

```
1 if not user_logged and not is_admin:
2     return
3
4 print("Admin panel")
```

TERNÁRNÍ OPERÁTOR

```
1 status = "OK" if value > 0 else "FAIL"
```

TERNÁRNÍ OPERÁTOR

```
1 status = "OK" if value > 0 else "FAIL"
```

```
1 n = -5
```

```
2
```

```
3 res = "Positive" if n > 0 else "Negative" if n < 0 else "Zero"
```

TERNÁRNÍ OPERÁTOR

```
1 status = "OK" if value > 0 else "FAIL"
```

```
1 n = -5
```

```
2
```

```
3 res = "Positive" if n > 0 else "Negative" if n < 0 else "Zero"
```

Používat jen pro jednoduché výrazy

PATTERN MATCHING (MATCH)

```
1 cmd = "start"
2
3 match cmd:
4     case "start":
5         run()
6     case "stop":
7         stop()
8     case _:
9         print("Unknown")
```

PATTERN MATCHING (MATCH)

```
1 cmd = "start"
2
3 match cmd:
4     case "start":
5         run()
6     case "stop":
7         stop()
8     case _:
9         print("Unknown")
```

```
1 x = 20
2
3 match x:
4     case 10 | 20 | 30: # Matches 10, 20, or 30
5         print(f"Matched: {x}")
6     case _:
7         print("No match found")
```

Andrej Pčelovodov

FOR CYKLUS (LIST)

Pomocí smyčky for můžeme provést sadu příkazů, jednou pro každou položku v:

list
tuple
set atd.

```
1 fruits = ["apple", "banana", "cherry"]  
2 for x in fruits:  
3     print(x)
```


FOR CYKLUS (LIST)

Pomocí smyčky for můžeme provést sadu příkazů, jednou pro každou položku v:

list
tuple
set atd.

```
1 fruits = ["apple", "banana", "cherry"]  
2 for x in fruits:  
3     print(x)
```

FOR CYKLUS (LIST)

Pomocí smyčky for můžeme provést sadu příkazů, jednou pro každou položku v:

list
tuple
set atd.

```
1 fruits = ["apple", "banana", "cherry"]  
2 for x in fruits:  
3     print(x)
```

FOR CYKLUS (LIST)

Pomocí smyčky for můžeme provést sadu příkazů, jednou pro každou položku v:

list
tuple
set atd.

```
1 fruits = ["apple", "banana", "cherry"]  
2 for x in fruits:  
3     print(x)
```

FOR CYKLUS (LIST)

Pomocí smyčky for můžeme provést sadu příkazů, jednou pro každou položku v:

list
tuple
set atd.

```
1 fruits = ["apple", "banana", "cherry"]  
2 for x in fruits:  
3     print(x)
```



FOR CYKLUS (LIST)

Pomocí smyčky for můžeme provést sadu příkazů, jednou pro každou položku v:

list
tuple
set atd.

```
1 fruits = ["apple", "banana", "cherry"]  
2 for x in fruits:  
3     print(x)
```



```
1 for name in ["Alice", "Bob", "Eve"]:  
2     print(name)
```

FOR CYKLUS (LIST)

Pomocí smyčky for můžeme provést sadu příkazů, jednou pro každou položku v:

list
tuple
set atd.

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     print(x)
```



```
1 for name in ["Alice", "Bob", "Eve"]:
2     print(name)
```



FOR CYKLUS (STRING)

I řetězce jsou iterovatelné objekty (obsahují posloupnost znaků)

```
1 fruit = "banana"
2
3 for x in fruit:
4     print(x)
```

FOR CYKLUS (STRING)

I řetězce jsou iterovatelné objekty (obsahují posloupnost znaků)

```
1 fruit = "banana"
2
3 for x in fruit:
4     print(x)
```


FOR CYKLUS (STRING)

I řetězce jsou iterovatelné objekty (obsahují posloupnost znaků)

```
1 fruit = "banana"
2
3 for x in fruit:
4     print(x)
```

FOR CYKLUS (STRING)

I řetězce jsou iterovatelné objekty (obsahují posloupnost znaků)

```
1 fruit = "banana"
2
3 for x in fruit:
4     print(x)
```

FOR CYKLUS (BREAK)

Pomocí příkazu `break` můžeme smyčku zastavit dříve, než projde všemi položkami

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     print(x)
4     if x == "banana":
5         break
```

FOR CYKLUS (BREAK)

Pomocí příkazu `break` můžeme smyčku zastavit dříve, než projde všemi položkami

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     print(x)
4     if x == "banana":
5         break
```

FOR CYKLUS (BREAK)

Pomocí příkazu `break` můžeme smyčku zastavit dříve, než projde všemi položkami

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     print(x)
4     if x == "banana":
5         break
```

FOR CYKLUS (BREAK)

Pomocí příkazu `break` můžeme smyčku zastavit dříve, než projde všemi položkami

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     print(x)
4     if x == "banana":
5         break
```

FOR CYKLUS (BREAK)

Pomocí příkazu `break` můžeme smyčku zastavit dříve, než projde všemi položkami

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     print(x)
4     if x == "banana":
5         break
```

FOR CYKLUS (BREAK)

Pomocí příkazu `break` můžeme smyčku zastavit dříve, než projde všemi položkami

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     print(x)
4     if x == "banana":
5         break
```


FOR CYKLUS (CONTINUE)

Pomocí příkazu `continue` můžeme zastavit aktuální iteraci smyčky a pokračovat další.

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     if x == "banana":
4         continue
5     print(x)
```

FOR CYKLUS (CONTINUE)

Pomocí příkazu `continue` můžeme zastavit aktuální iteraci smyčky a pokračovat další.

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     if x == "banana":
4         continue
5     print(x)
```

FOR CYKLUS (CONTINUE)

Pomocí příkazu `continue` můžeme zastavit aktuální iteraci smyčky a pokračovat další.

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     if x == "banana":
4         continue
5     print(x)
```

FOR CYKLUS (CONTINUE)

Pomocí příkazu `continue` můžeme zastavit aktuální iteraci smyčky a pokračovat další.

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     if x == "banana":
4         continue
5     print(x)
```

FOR CYKLUS (CONTINUE)

Pomocí příkazu `continue` můžeme zastavit aktuální iteraci smyčky a pokračovat další.

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     if x == "banana":
4         continue
5     print(x)
```

FOR CYKLUS (CONTINUE)

Pomocí příkazu `continue` můžeme zastavit aktuální iteraci smyčky a pokračovat další.

```
1 fruits = ["apple", "banana", "cherry"]
2 for x in fruits:
3     if x == "banana":
4         continue
5     print(x)
```

FOR CYKLUS (RANGE)

Chcete-li provést smyčku přes sadu kódu určitý početkrát, můžete použít funkci `range()`. Funkce `range()` vrací posloupnost čísel, která začíná standardně od 0, zvyšuje se o 1 (standardně) a končí u zadaného čísla.

```
1 for x in range(6):  
2     print(x)
```

FOR CYKLUS (RANGE)

Chcete-li provést smyčku přes sadu kódu určitý početkrát, můžete použít funkci `range()`.
Funkce `range()` vrací posloupnost čísel, která začíná standardně od 0, zvyšuje se o 1 (standardně) a končí u zadaného čísla.

```
1 for x in range(6):  
2     print(x)
```


FOR CYKLUS (RANGE)

Chcete-li provést smyčku přes sadu kódu určitý početkrát, můžete použít funkci `range()`. Funkce `range()` vrací posloupnost čísel, která začíná standardně od 0, zvyšuje se o 1 (standardně) a končí u zadaného čísla.

```
1 for x in range(6):  
2     print(x)
```

RANGE()

Funkce `range()` má výchozí počáteční hodnotu 0, ale je možné zadat počáteční hodnotu přidáním parametru: `range(1, 5)`, což znamená hodnoty od 1 do 5 (ale bez 5).

Funkce `range()` ve výchozím nastavení zvyšuje posloupnost o 1, je však možné zadat hodnotu přírůstku přidáním třetího parametru: `range(0, 10, 2)`:

```
1 range(5)           # 0..4
2 range(1, 5)        # 1..4
3 range(0, 10, 2)    # krok 2
```

RANGE()

Funkce `range()` má výchozí počáteční hodnotu 0, ale je možné zadat počáteční hodnotu přidáním parametru: `range(1, 5)`, což znamená hodnoty od 1 do 5 (ale bez 5).

Funkce `range()` ve výchozím nastavení zvyšuje posloupnost o 1, je však možné zadat hodnotu přírůstku přidáním třetího parametru: `range(0, 10, 2)`:

```
1 range(5)           # 0..4
2 range(1, 5)        # 1..4
3 range(0, 10, 2)    # krok 2
```

RANGE()

Funkce `range()` má výchozí počáteční hodnotu 0, ale je možné zadat počáteční hodnotu přidáním parametru: `range(1, 5)`, což znamená hodnoty od 1 do 5 (ale bez 5).

Funkce `range()` ve výchozím nastavení zvyšuje posloupnost o 1, je však možné zadat hodnotu přírůstku přidáním třetího parametru: `range(0, 10, 2)`:

```
1 range(5)           # 0..4
2 range(1, 5)        # 1..4
3 range(0, 10, 2)    # krok 2
```

RANGE()

Funkce `range()` má výchozí počáteční hodnotu 0, ale je možné zadat počáteční hodnotu přidáním parametru: `range(1, 5)`, což znamená hodnoty od 1 do 5 (ale bez 5).

Funkce `range()` ve výchozím nastavení zvyšuje posloupnost o 1, je však možné zadat hodnotu přírůstku přidáním třetího parametru: `range(0, 10, 2)`:

```
1 range(5)           # 0..4
2 range(1, 5)        # 1..4
3 range(0, 10, 2)    # krok 2
```

FOR CYKLUS (ELSE)

Klíčové slovo `else` ve smyčce `for` určuje blok kódu, který se má provést po dokončení smyčky.

Blok `else` NEBUDE proveden, pokud je smyčka zastavena příkazem `break`.

```
1 for x in range(6):  
2     if x == 3: break  
3     print(x)  
4 else:  
5     print("Finally finished!")
```

FOR CYKLUS (ELSE)

Klíčové slovo `else` ve smyčce `for` určuje blok kódu, který se má provést po dokončení smyčky.

Blok `else` NEBUDE proveden, pokud je smyčka zastavena příkazem `break`.

```
1 for x in range(6):  
2     if x == 3: break  
3     print(x)  
4 else:  
5     print("Finally finished!")
```

FOR CYKLUS (ELSE)

Klíčové slovo `else` ve smyčce `for` určuje blok kódu, který se má provést po dokončení smyčky.

Blok `else` NEBUDE proveden, pokud je smyčka zastavena příkazem `break`.

```
1 for x in range(6):  
2     if x == 3: break  
3     print(x)  
4 else:  
5     print("Finally finished!")
```


FOR CYKLUS (ELSE)

Klíčové slovo `else` ve smyčce `for` určuje blok kódu, který se má provést po dokončení smyčky.

Blok `else` NEBUDE proveden, pokud je smyčka zastavena příkazem `break`.

```
1 for x in range(6):  
2     if x == 3: break  
3     print(x)  
4 else:  
5     print("Finally finished!")
```

FOR CYKLUS (ELSE)

Klíčové slovo `else` ve smyčce `for` určuje blok kódu, který se má provést po dokončení smyčky.

Blok `else` NEBUDE proveden, pokud je smyčka zastavena příkazem `break`.

```
1 for x in range(6):  
2     if x == 3: break  
3     print(x)  
4 else:  
5     print("Finally finished!")
```

FOR CYKLUS (ELSE)

Klíčové slovo `else` ve smyčce `for` určuje blok kódu, který se má provést po dokončení smyčky.

Blok `else` NEBUDE proveden, pokud je smyčka zastavena příkazem `break`.

```
1 for x in range(6):  
2     if x == 3: break  
3     print(x)  
4 else:  
5     print("Finally finished!")
```

VNOŘENÝ FOR CYKLUS

Vnořená smyčka je smyčka uvnitř smyčky.

„Vnitřní smyčka“ se provede jednou za každou iteraci „vnější smyčky“:

```
1 adj = ["red", "big", "tasty"]
2 fruits = ["apple", "banana", "cherry"]
3
4 for x in adj:
5     for y in fruits:
6         print(x, y)
```

VNOŘENÝ FOR CYKLUS

Vnořená smyčka je smyčka uvnitř smyčky.

„Vnitřní smyčka“ se provede jednou za každou iteraci „vnější smyčky“:

```
1 adj = ["red", "big", "tasty"]
2 fruits = ["apple", "banana", "cherry"]
3
4 for x in adj:
5     for y in fruits:
6         print(x, y)
```

VNOŘENÝ FOR CYKLUS

Vnořená smyčka je smyčka uvnitř smyčky.

„Vnitřní smyčka“ se provede jednou za každou iteraci „vnější smyčky“:

```
1 adj = ["red", "big", "tasty"]
2 fruits = ["apple", "banana", "cherry"]
3
4 for x in adj:
5     for y in fruits:
6         print(x, y)
```

VNOŘENÝ FOR CYKLUS

Vnořená smyčka je smyčka uvnitř smyčky.

„Vnitřní smyčka“ se provede jednou za každou iteraci „vnější smyčky“:

```
1 adj = ["red", "big", "tasty"]
2 fruits = ["apple", "banana", "cherry"]
3
4 for x in adj:
5     for y in fruits:
6         print(x, y)
```

VNOŘENÝ FOR CYKLUS

Vnořená smyčka je smyčka uvnitř smyčky.

„Vnitřní smyčka“ se provede jednou za každou iteraci „vnější smyčky“:

```
1 adj = ["red", "big", "tasty"]
2 fruits = ["apple", "banana", "cherry"]
3
4 for x in adj:
5     for y in fruits:
6         print(x, y)
```


VNOŘENÝ FOR CYKLUS

Vnořená smyčka je smyčka uvnitř smyčky.

„Vnitřní smyčka“ se provede jednou za každou iteraci „vnější smyčky“:

```
1 adj = ["red", "big", "tasty"]
2 fruits = ["apple", "banana", "cherry"]
3
4 for x in adj:
5     for y in fruits:
6         print(x, y)
```

PASS

```
1 for x in range(100):  
2     pass
```

Příkaz `pass` se používá jako zástupný symbol pro budoucí kód.

Při provedení příkazu `pass` se nic nestane,
ale zabráníte tak chybě, když není povolen prázdný kód.

Prázdný kód není povolen ve:

- smyčkách**
- definicích funkcí**
- definicích tříd**
- v příkazech `if`**

PASS

```
1 for x in range(100):  
2     pass
```

Příkaz `pass` se používá jako zástupný symbol pro budoucí kód.

Při provedení příkazu `pass` se nic nestane,
ale zabráníte tak chybě, když není povolen prázdný kód.

Prázdný kód není povolen ve:

- smyčkách**
- definicích funkcí**
- definicích tříd**
- v příkazech `if`**

PASS

```
1 for x in range(100):  
2     pass
```

Příkaz `pass` se používá jako zástupný symbol pro budoucí kód.

Při provedení příkazu `pass` se nic nestane,
ale zabráníte tak chybě, když není povolen prázdný kód.

Prázdný kód není povolen ve:

- smyčkách**
- definicích funkcí**
- definicích tříd**
- v příkazech `if`**

PASS

```
1 def myfunction():  
2     pass
```

```
1 class Person:  
2     pass
```

```
1 a = 33  
2 b = 200  
3  
4 if b > a:  
5     pass
```

SMYČKA WHILE

Pomocí smyčky while můžeme provádět sadu příkazů, dokud je podmínka pravdivá.

```
1 i = 1
2 while i < 6:
3     print(i)
4     i += 1
```

Smyčka while vyžaduje, aby byly připraveny příslušné proměnné.

V tomto příkladu musíme definovat indexovací proměnnou i, kterou nastavíme na hodnotu 1.

SMYČKA WHILE

Pomocí smyčky while můžeme provádět sadu příkazů, dokud je podmínka pravdivá.

```
1 i = 1
2 while i < 6:
3     print(i)
4     i += 1
```

Smyčka while vyžaduje, aby byly připraveny příslušné proměnné.

V tomto příkladu musíme definovat indexovací proměnnou i, kterou nastavíme na hodnotu 1.

Poznámka: nezapomeňte zvyšovat hodnotu i, jinak bude smyčka pokračovat donekonečna.

SMYČKA WHILE (BREAK)

Pomocí příkazu `break` můžeme smyčku zastavit, i když je podmínka `while` splněna.

```
1 i = 1
2 while i < 6:
3     print(i)
4     if i == 3:
5         break
6     i += 1
```


SMYČKA WHILE (CONTINUE)

Pomocí příkazu continue
můžeme zastavit aktuální iteraci a pokračovat další.

```
1 i = 0
2 while i < 6:
3     i += 1
4     if i == 3:
5         continue
6     print(i)
```

SMYČKA WHILE (ELSE)

Pomocí příkazu else můžeme spustit blok kódu jednou, když podmínka již není splněna.

```
1 i = 1
2 while i < 6:
3     print(i)
4     i += 1
5 else:
6     print("i is no longer less than 6")
```

WALRUS OPERÁTOR :=



WALRUS OPERÁTOR :=

WALRUS OPERÁTOR :=

```
1 while (line := file.readline()):  
2     print(line)
```

WALRUS OPERÁTOR :=

```
1 d = [  
2     {"userId": 1, "name": "rahul", "completed": False},  
3     {"userId": 1, "name": "rohit", "completed": False},  
4     {"userId": 1, "name": "ram", "completed": False},  
5     {"userId": 1, "name": "ravan", "completed": True}  
6 ]  
7  
8 print("With Python 3.8 Walrus Operator:")  
9 for entry in d:  
10     if name := entry.get("name"):  
11         print(name)  
12  
13 print("Without Walrus operator:")  
14 for entry in d:  
15     name = entry.get("name")
```

ITERABLE VS ITERATOR

Seznamy (list), tuple, slovníky (dict) a množiny (set) jsou všechny iterovatelné objekty.

Jedná se o iterovatelné kontejnery, ze kterých můžete získat iterátor.

Všechny tyto objekty mají metodu `iter()`, která se používá k získání iterátoru

ITERABLE VS ITERATOR

Seznamy (list), tuple, slovníky (dict) a množiny (set) jsou všechny iterovatelné objekty.

Jedná se o iterovatelné kontejnery, ze kterých můžete získat iterátor.

Všechny tyto objekty mají metodu `iter()`, která se používá k získání iterátoru

- Iterable → lze projít (list, str, dict)

ITERABLE VS ITERATOR

Seznamy (list), tuple, slovníky (dict) a množiny (set) jsou všechny iterovatelné objekty.

Jedná se o iterovatelné kontejnery, ze kterých můžete získat iterátor.

Všechny tyto objekty mají metodu `iter()`, která se používá k získání iterátoru

- Iterable → lze projít (list, str, dict)
- Iterator → má `__next__()`

ITERABLE VS ITERATOR

```
1 mytuple = ("apple", "banana", "cherry")
2 myit = iter(mytuple)
3
4 print(next(myit))
5 print(next(myit))
6 print(next(myit))
```

ITERABLE VS ITERATOR

I řetězce jsou iterovatelné objekty a mohou vrátit iterátor

```
1 mystr = "banana"
2 myit = iter(mystr)
3
4 print(next(myit))
5 print(next(myit))
6 print(next(myit))
7 print(next(myit))
8 print(next(myit))
9 print(next(myit))
```

PROCHÁZENÍ ITERÁTOREM

Můžeme také použít smyčku for k iterování přes iterovatelný objekt

```
1 mytuple = ("apple", "banana", "cherry")
2
3 for x in mytuple:
4     print(x)
```

```
1 mystr = "banana"
2
3 for x in mystr:
4     print(x)
```

VÍCE SEKVENCÍ NAJEDNOU

```
1 names = ["A", "B"]
2 scores = [10, 20]
3
4 for n, s in zip(names, scores):
5     print(n, s)
```

ITERTOOLS

ITERTOOLS

- chain

ITERTOOLS

- chain
- count

ITERTOOLS

- chain
- count
- cycle

ITERTOOLS

- chain
- count
- cycle
- islice

KOMBINATORICKÉ GENERÁTORY

```
1 from itertools import permutations, combinations
2
3 permutations([1,2,3])
4 combinations([1,2,3], 2)
```

SHRNUTÍ

SHRNUTÍ

- if / match → rozhodování

SHRNUTÍ

- if / match → rozhodování
- for / while → opakování

SHRNUTÍ

- if / match → rozhodování
- for / while → opakování
- itertools → výkon a elegance