LAST LESSON

FOR

Pro každý prvek x z collection prováděj kód

```
for x in collection:
        process(x)
```

FOR EXAMPLE

```
>>> for x in ('alfa', 'beta', 'gamma'):
...     print(x)
...
alfa
beta
gamma
```

FOR

```
for x in 'abc':
```

```
for x in ['alfa', 'beta', 'gamma']:
```

```
for x in {'alfa', 'beta', 'gamma'}:
```

```
for x in {'alfa': 0, 'beta': 1, 'gamma': 2}:
```

RANGE

```
for x in (0, 1, 2, 4, 5, 6, 7, 8, 9):
        print(x)
```

```
for x in range(10):
        print(x)
```

RANGE

```
range(stop)
range(start, stop)
range(start, stop, step)
```

```
range(5)
```

```
0, 1, 2, 3, 4
```

```
range(5, 10)
```

```
5, 6, 7, 8, 9
```

```
range(5, 10, 2)
```

```
5, 7, 9
```

```
range(10, 5, -1)
```

```
10, 9, 8, 7, 6
```

TODAY'S LESSON

FUNCTION

- Named block of code
- Accepts arguments
- Can return result

FUNCTION

```python
def add_numbers(number_1, number_2):
    result = number_1 + number_2
    return result
```

```
>>> add_numbers(4, 6)
10
>>> add_numbers(1, 2)
3
```

FUNCTION WITHOUT RETURN

```
def add_year(data_dict, year):
    data_dict['year'] = year
```

```
>>> data = {'movie': 'Joker'}
>>> add_year(data, 2019)
>>> data
{'movie': 'Joker', 'year': 2019}
```

FUNCTION WITHOUT ARGUMENTS AND WITHOUT RETURN

```python
def print_separator():
    print('-' * 80)
```

MORE COMPLEX FUNCTION

```
def sum_positive(numbers):
    total = 0
    for number in numbers:
        if number > 0:
            total += number
    return total
```

```
>>> sum_positive([1, 0, -5, 4, -10, 3])
8
```

ERROR IN FUNCTION

```python
def sum_positive(numbers):
    total = 0
    for number in numbers:
        if number > 0:
            total += number
    return total

number = 42
sum_positive(number)
```

```
$ python3 error_in_function.py
Traceback (most recent call last):
  File "error_in_function.py", line 9, in <module>
    sum_positive(number)
  File "error_in_function.py", line 3, in sum_positive
    for number in numbers:
TypeError: 'int' object is not iterable
```

EXERCISES

SUM_RANGE

1. Arguments: start, stop
2. Return the sum of all numbers in range between start and stop (stop including)

```
>>> sum_range(0, 5)
15
>>> sum_range(5, 10)
45
```

IS_PRIME

1. Arguments: number
2. Return True if number is prime False otherwise

```
>>> is_prime(1)
True
>>> is_prime(2)
True
>>> is_prime(3)
True
>>> is_prime(4)
False
>>> is_prime(5)
True
>>> is_prime(6)
False
```

SUM_PRIME_RANGE

1. Arguments: start, stop
2. Return the sum of all primes in range between start and stop (stop including)

```
>>> sum_prime_range(5, 10)
12
>>> sum_predicate_range(5, 11)
23
```

FUNCTION AS AN OBJECT

```
>>> def add(a, b):
...     return a + b

>>> new_add = add
>>> new_add(4, 5)
9
>>> add(4, 5)
9
```

SUM_PREDICATE_RANGE

1. Arguments: start, stop, predicate
2. Return the sum of all numbers between start and stop for which predicate is True

```
>>> sum_predicate_range(5, 10, is_prime)
12
>>> sum_predicate_range(5, 11, is_prime)
23
>>> def is_divisible_by_2(number):
...     return number % 2 == 0
>>> sum_predicate_range(5, 10, is_divisible_by_2)
24
```

END?

COLLECTION ASSIGNMENT

```
>>> a, b, c = (1, 2, 3)
>>> a
1
>>> b
2
>>> c
3
```

```
>>> a, *b = (1, 2, 3)
>>> a
1
>>> b
[2, 3]
```

```
>>> a, *b, c = (1, 2, 3, 4, 5)
>>> a
1
>>> b
[2, 3, 4]
>>> c
5
```

IMPORT

```
>>> import random
>>> random.random()
0.6914667260222296
>>> random.randint(10, 20)
17
>>> import string
>>> random.choice(string.ascii_letters)
'Q'
```

https://docs.python.org/3/library/random.html

https://docs.python.org/3/library/string.html

SPECIAL CHARACTERS

- '\n' - newline
- '\t' - tabulator
- '\r' - carriage return

```
>>> print('This is the first line\nAnd this is the second line')
This is the first line
And this is the second line
>>> print('This is just symbol backslash \\')
This is just symbol backslash \
```

LIST COMPREHENSION

```
powers = []
for x in range(10):
        powers.append(x ** 2)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
powers = [x ** 2 for x in range(10)]
```

DICT COMPREHENSION

```python
powers = {}
for x in range(10):
        powers[x] = x ** 2
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81
```

```python
powers = {x: x ** 2 for x in range(10)}
```