Basic Programming IV: Functions

DigitalSkola
Uncover The World Of Digital Skills With Us

Table of Content

What will We Learn Today?

1. Functional Programming Concept
2. Let's Create Functions
3. Introduction to Library
4. String Manipulation

DigitalSkola
Uncover The World Of Digital Skills With Us

Quote of The Day

“Any person can write code that a computer can understand. Good programmers write code that humans can understand.”

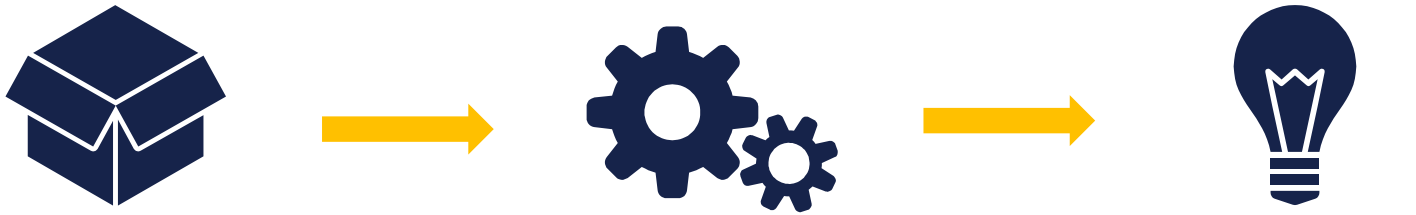
- **Martin Fowler**

DigitalSkola
Uncover The World Of Digital Skills With Us

What is functional programming?

Functional Programming is a programming paradigm with software primarily composed of functions processing data throughout its execution.

To put it simply, function is like a factory which has input, sequence of processes, and output.



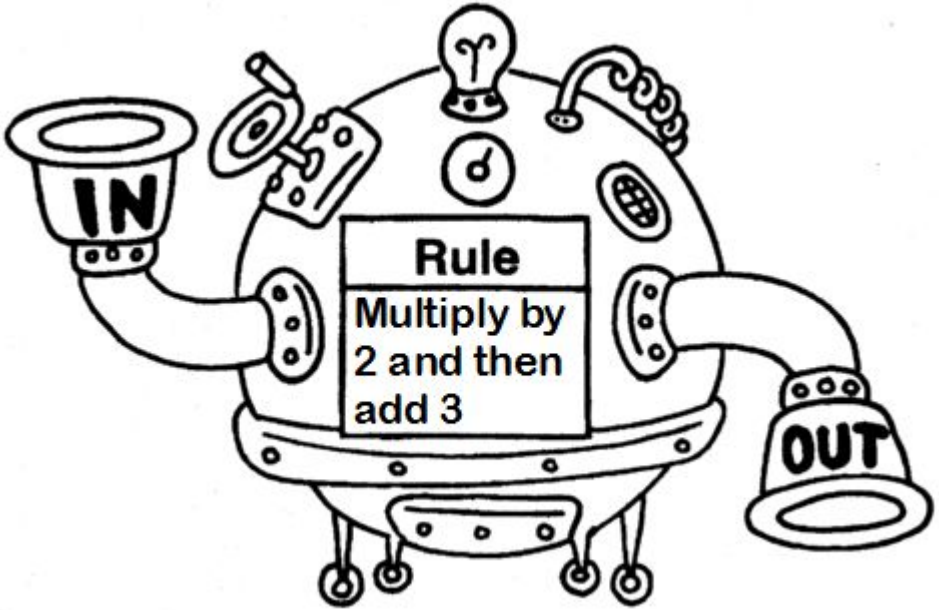
DigitalSkola
Uncover The World Of Digital Skills With Us

What we will agree on:

- **In Scope**
  - **Functions**
    - Why it is **useful**?
    - Deep dive on **basic** functions
  - **Best practices**
    - How to make it **user-friendly**
- **Out Scope**
  - Complex function
    - You will get used to it along the way! :D

DigitalSkola
Uncover The World Of Digital Skills With Us

Function helps your repetitive work!



Built-in function

- str()

```
x = str(5)

print(x)
```

```
'5'
```

```
print(type(x))
```

```
<class 'str'>
```

Defining a function

```
def square():     # <- Function header
    new_value = 4 ** 2     # <- Function body
    print(new_value)
square()
```

```
16
```

Function parameters

```
def square(value):
    new_value = value ** 2
    print(new_value)

square(4)
```

```
16
```

```
square(5)
```

```
25
```

DigitalSkola
Uncover The World Of Digital Skills With Us

Component of Function

Defining a function started with def, name, parentheses “()”, colon “:”.

Every process inside function must be in tab or 4 spaces. We must choose between one of them. Because Python is space sensitive. Several services like Google Cloud Platform prefer 4 spaces to define function.

Input

```
def wealth_estimator(money):
    if money >= 1000000:
        status = 'You are above 1%'

    elif money < 1000000 and money <= 100:
        status = 'You are in middle class'

    else:
        status = 'You are poor'

    return status
```

Process

Output

Input

- In Python functions, **each input is called parameter**. So, when 'parameter' is mentioned, keep in mind that it is input of a function.
- We can put any data type inside parameter, for example string, integer, float, array, dictionary, even function itself.
- Regardless the flexibility of Python's parameter, once it is defined as a certain data type, do not insert it with different with other type. If we originally define parameter 'name' as a string, then do not insert it with integer (unless it is integer to float).

Input (2)

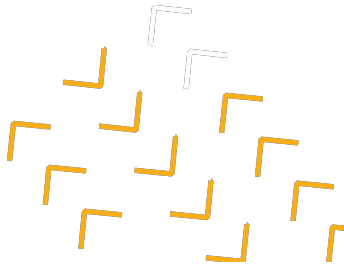
- Parameter is a variable inside the function that cannot be called from outside function, unless we put it as an output.
- A function also can have predefined or default parameter. If we do not insert any value, it will call its default value. Position of predefined parameter must be after empty one because it is not allowed to be empty.

Process

- Process inside function is like a sequence of steps to achieve something (such as output).
- Local variable inside function can only stay within it and will be deleted afterwards.
- We can do almost anything inside function, such as looping, conditional statement, or even call other function.
- Any process from any library can also work inside function.
- Make sure to give comment in each process to remind your future self or your teammates.

Output

- Output of function can be message or value. It depends on how we define it.
- To produce message, we can use **print**.
- To produce values, we can use **return**.
- Python function can produce multiple values at the same time.
- If we want to ignore certain value from multiple **returns**, use underscore (_).



DigitalSkola
Uncover The World Of Digital Skills With Us

What's the output?

```
def shout(word):
    shout_word = word + '!!!'
    print(shout_word)

shout('congratulations')
```

Congratulations 👍

Congratulations!!! 👏

Return value from functions

```
def square(value):
    new_value = value ** 2
    return new_value
num = square(4)

print(num)
```

```
16
```

Which one is better?

```
def square(value):
    new_value = value ** 2
    return new_value
```

```
def square(value):
    """Return the square of a value."""
    new_value = value ** 2
    return new_value
```

Docstrings

```
def square(value):
    new_value = value ** 2
    return new_value
```

```
def square(value):
    """Return the square of a value."""
    new_value = value ** 2
    return new_value
```

- **Describe** what function does
- Serve as **documentation**
- Placed in immediate line **after header**

Multiple function parameters

```
def raise_to_power(value1, value2):
    """Raise value1 to the power of value2."""
    new_value = value1 ** value2
    return new_value
```

DigitalSkola
Uncover The World Of Digital Skills With Us

Benefit of Function

1. When you have repetitive process, you can use function to simplify the code.
2. Separate every workflow to its function.
   (E.g., Clean data, Call data from csv, etc.)
3. Having function in code will ease us to focus on solving problem, instead of debugging same process again and again.
4. Each problem or error is easy to trace through function.
5. Relatively useful to reduce memory usage.

Scope

- **Not all objects** are accessible everywhere in the script
- **Scope:** part where object/name may be accessible
  - **Global** → defined in the main body of the script
  - **Local** → inside a function
  - **Built-in** → names in the pre-defined built-ins module

Global vs. Local Variable

- **Global variable** can be accessed throughout the program body by all functions/class. It stays inside memory, unless we delete it or close the Python program.
- **Local variable** can be accessed only inside the function in which they are declared. After the function completed its work, the local variable will be automatically deleted.

Global vs Local Scope (1)

```
def square(value):
    """Returns the square of a number."""
    new_val = value ** 2
    return new_val
square(3)
```

```
9
```

```
new_val
```

```
<hr />---------------------------------------------------------------
NameError                       Traceback (most recent call last)
<ipython-input-3-3cc6c6de5c5c> in <module>()
<hr />-> 1 new_value
NameError: name 'new_val' is not defined
```

Global vs Local Scope (2)

```
new_val = 10

def square(value):
    """Returns the square of a number."""
    new_val = value ** 2
    return new_val
square(3)
```

```
9
```

```
new_val
```

```
10
```

Global vs Local Scope (3)

```
new_val = 10

def square(value):
    """Returns the square of a number."""
    new_value2 = new_val ** 2
    return new_value2
square(3)
```

```
100
```

```
new_val = 20

square(3)
```

```
400
```

Global vs Local Scope (4)

```
new_val = 10

def square(value):
    """Returns the square of a number."""
    global new_val
    new_val = new_val ** 2
    return new_val
square(3)
```

```
100
```

```
new_val
```

```
100
```

Global vs Local Scope (4)

```
def func1():
    num = 3
    print(num)

def func2():
    global num
    double_num = num * 2
    num = 6
    print(double_num)
```

- What are the values of func1() and func2()?
- What is the value of num after calling both functions?

Nested Function (1)

```
def outer( ... ):
    """ ... """
    x = ...

    def inner( ... ):
        """ ... """
        y = x ** 2
    return ...
```

Nested Function (2)

```
def mod2plus5(x1, x2, x3):
    """Returns the remainder plus 5 of three values."""

    new_x1 = x1 % 2 + 5
    new_x2 = x2 % 2 + 5
    new_x3 = x3 % 2 + 5

    return (new_x1, new_x2, new_x3)
```

Nested Function (3)

```
def mod2plus5(x1, x2, x3):
    """Returns the remainder plus 5 of three values."""

    def inner(x):
        """Returns the remainder plus 5 of a value."""
        return x % 2 + 5

    return (inner(x1), inner(x2), inner(x3))
```

```
print(mod2plus5(1, 2, 3))
```

Returning Function

```
def raise_val(n):
    """Return the inner function."""

    def inner(x):
    """Raise x to the power of n."""
        raised = x ** n
        return raised

    return inner
```

```
square = raise_val(2)
cube = raise_val(3)
print(square(2), cube(4))
```

```
4 64
```

Using Non-local

```
def outer():
    """Prints the value of n."""
    n = 1

    def inner():
        nonlocal n
        n = 2
        print(n)

    inner()
    print(n)
```

Scopes searched: LEGB rule

- Local scope
- Enclosing functions (if any)
- Global
- Built-in

Default Argument

```
def power(number, pow=1):
   """Raise number to the power of pow."""
   new_value = number ** pow
   return new_value

power(9, 2)
```

```
81
```

```
power(9, 1)
```

```
9
```

```
power(9)
```

```
9
```

DigitalSkola
Uncover The World Of Digital Skills With Us

Flexible Argument

```python
def add_all(*args):
    """Sum all values in *args together."""

    # Initialize sum
    sum_all = 0

    # Accumulate the sum
    for num in args:
        sum_all += num

    return sum_all
```

```python
add_all(1)
```

```
1
```

```python
add_all(1, 2)
```

```
3
```

```python
add_all(5, 10, 15, 20)
```

```
50
```

Flexible more than one type arguments

```
def print_all(**kwargs):
    """Print out key-value pairs in **kwargs."""

    # Print out the key-value pairs
    for key, value in kwargs.items():
        print(key + \": \" + value)
```

```
print_all(name="dumbledore", job="headmaster")
```

```
job: headmaster
name: dumbledore
```

Lambda Functions

```
raise_to_power = lambda x, y: x ** y

raise_to_power(2, 3)
```

```
8
```

Function Map

```
nums = [48, 6, 9, 21, 1]

square_all = map(lambda num: num ** 2, nums)

print(square_all)
```

```
<map object at 0x103e065c0>
```

```
print(list(square_all))
```

```
[2304, 36, 81, 441, 1]
```

- Takes **two** argument (func, seq)
- Applies the function to **ALL** element in the sequence

Error handling: Incorrect argument

```
float(2)
```

```
2.0
```

```
float('2.3')
```

```
2.3
```

```
float('hello')
```

```
<hr />---------------------------------------------------------------
ValueError                          Traceback (most recent call last)
<ipython-input-3-d0ce8bccc8b2> in <module>()
<hr />-> 1 float('hi')
ValueError: could not convert string to float: 'hello'
```

Errors and Exception

- **Exceptions** - caught during execution
- Catch exceptions with try-except clause
  - Runs the code following try
  - If there's an exception, run the code following except

DigitalSkola
Uncover The World Of Digital Skills With Us

Errors and Exception

```
def sqrt(x):
    """Returns the square root of a number."""
    try:
        return x ** 0.5
    except:
        print('x must be an int or float')

sqrt(4)
```

```
2.0
```

```
sqrt(10.0)
```

```
3.1622776601683795
```

```
sqrt('hi')
```

```
x must be an int or float
```

```
def sqrt(x):
    """Returns the square root of a number."""
    try:
        return x ** 0.5
    except TypeError:
        print('x must be an int or float')
```

Errors and Exception

```
sqrt(-9)
```

```
(1.8369701987210297e-16+3j)
```

```
def sqrt(x):
    """Returns the square root of a number."""
    if x < 0:
        raise ValueError('x must be non-negative')
    try:
        return x ** 0.5
    except TypeError:
        print('x must be an int or float')
```

```
sqrt(-2)
```

```
---------------------------------------------------------------
ValueError                          Traceback (most recent call last)
<ipython-input-2-4cf32322fa95> in <module>()
----> 1 sqrt(-2)
<ipython-input-1-a7b8126942e3> in sqrt(x)
      1 def sqrt(x):
      2     if x < 0:
----> 3         raise ValueError('x must be non-negative')
      4     try:
      5         return x**(0.5)
ValueError: x must be non-negative
```

Introduction to Library

What makes Python is a strong language for Data Science because of its libraries. A library is an outside code that we import to our project. We can use multiple libraries at the same time.

Library is a collection of code that is created by a person, community, or company. There are popular libraries for Data Science, such as numpy, pandas, sql-alchemy, matplotlib, seaborn, scikit-learn, tensorflow, etc.

By using library, almost 80% of our work is finished, and we can focus on the problem solving. We can create library for ourselves or contribute it for everyone around the world.

String Manipulation

Sometimes, data is provided in dirty text. This requires an advance methods solely specialized in language: **Natural Language Processing**. It gains insight from text and even voice.

- Slice
- Length
- Concatenate
- Lower case, Upper case, Capitalize
- Replace

DigitalSkola

Uncover The World Of Digital Skills With Us

Thank YOU