HACKTIV8

Python: Functions and Module

FTDS Phase 0 - Week 1 Day 3 PM

Contents

Introduction to

Basic Python Function

A function is like a set of instructions that does a specific job.

For example, imagine you want to make a program that draws a circle and then colors it. To solve this problem, you can create two functions:

1. The "create a circle" function, which draws the circle.
2. The "color" function, which adds color to the circle.

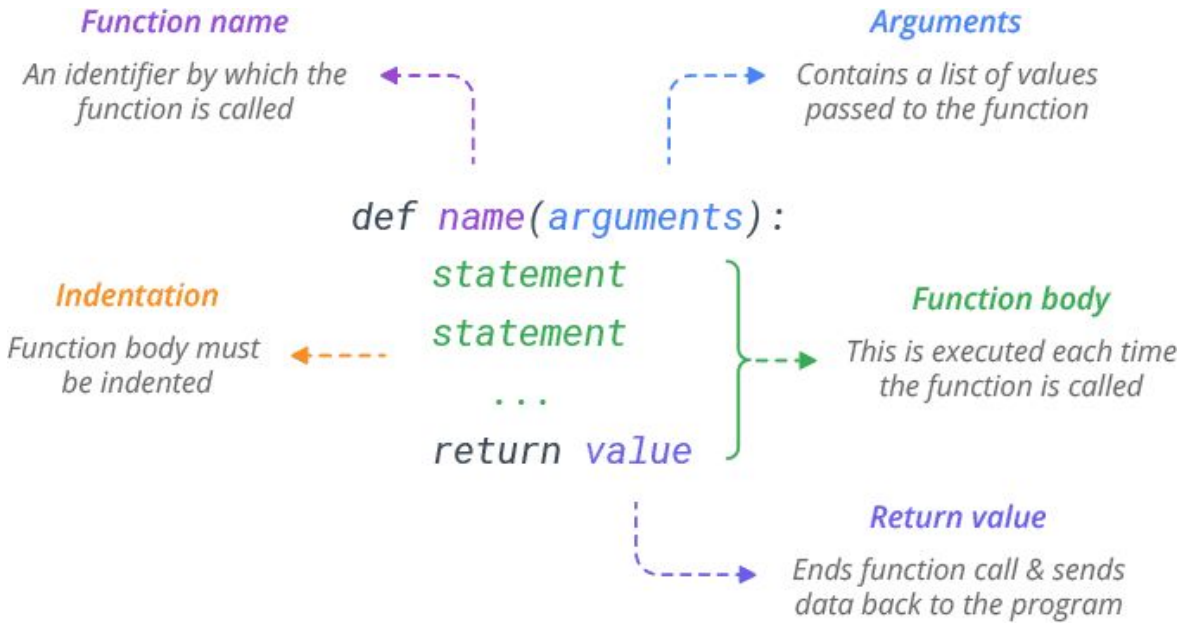
When we break down a big problem into smaller parts like this, it's easier to understand and reuse our program. To declare a function, the syntax will be:

```
def function_name(arguments):
    # function body

    return
```

Basic Python Function

Defining a Function

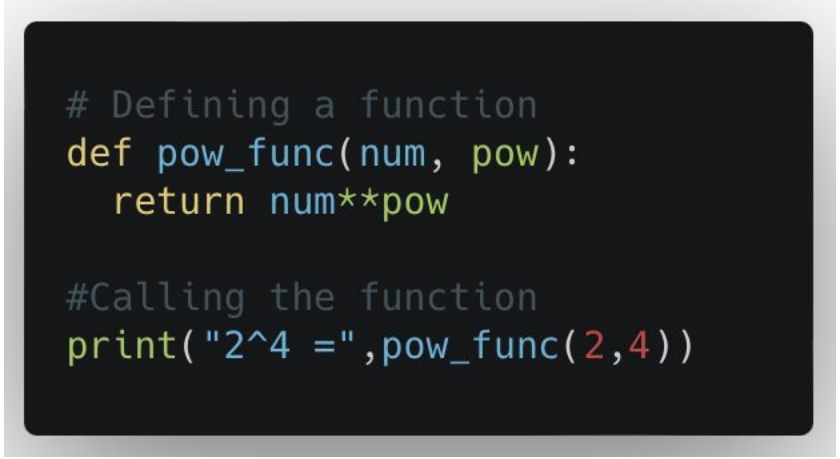


Courtesy:

https://www.learnbyexample.org/python-functions/

Basic Python Function

Calling Function



Let you have defined a function to calculate the exponential that is number power of another number.

Then, to use your constructed function, you can just call the function name **pow_func** followed by parentheses and input the value that you want to replace on **num** and **pow**.

Basic Python Function

Global vs Local Variable

```
c = 1 #Global variable
def func(x1):
  L = x1**2 + c #, L -> Local variable
  print(L)
```

In python function there is a concept that if you define variables in the function, you cannot call the variable outside the function.

The variables called **Local Variables**. However, if you have variables that exist in your general code and the can be used on the function. Such variables are called **Global Variables**.

```
L
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-a9dfae9159c1> in <module>
----> 1 L

NameError: name 'L' is not defined
```

If you call the variable L, it comes an error which the variable is not defined. So, L is local variable that only exist in the function.

Function Arguments

Required, Keyword, and Default Argument

Arguments are value that accepted by a function. What kind of value that be accepted? There are four kinds of arguments which are required, keyword, default, and variable-length argument.

Required Argument

Arguments that should have to be filled when the function is called.

Default Argument

Arguments that have already filled by default, and it's optional to fill the argument when the function is called

```
def func(a, b, c=1, d=3):
  return a+b+c+d
```

Potitional Argument

1 and 2 are belong to argument a and b respectively. The value will be filled to the arguments based on the order.

Keyword Argument

To use the keyword argument, you have to call the certain arguments that want to be filld by a value. The order is not matter

```
func(1, 2, d=3)
```

Function Arguments

Variable-Length Argument - Non Keyword

Variable-length argument is a unique argument in python function. It accept unlimited input of data. Therefore, the inputs will be saved to **Tuple (Non-keyword arguments denoted as *args)** or Dictionary **(Keyword arguments denoted as **kwargs)**

```
def multiplier(*num):
    prod = 1
#initialize prod variable with zero
    for i in num:
        prod = prod * i

    print("Product:",prod)

multiplier(3,5)     #Product: 15
multiplier(1,2,4)   #Product: 8
multiplier(2,2,6,7) #Product(168)
```

***args** can be change to any variable name sync as ***num** in example above.

Function Arguments

Variable-Length Argument - Keyword

```
def daftar_nama(sekolah, **nama_peserta):
  print(f"Peserta didik sekolah {sekolah}:")
  i = 0
  for nama in nama_peserta.values():
    i += 1
    print(f"{i}. {nama}")
  print(nama_peserta)

daftar_nama("Hacktiv8",siswa1="A",siswa2="B",siswa3="C",siswa4="D")

'''
_Output_

Peserta didik sekolah Hacktiv8:
1. A
2. B
3. C
4. D
{'siswa1': 'A', 'siswa2': 'B', 'siswa3': 'C', 'siswa4': 'D'}
'''
```

“Hacktiv8” will be inserted to sekolah variable and “A”, ”B”, ”C”, “D” will belong to nama_peserta which is a dictionary that consist of four keywords regarding the input that accepted by the function.

Introduction to

Anonymous Function

Anonymous function is a function that has no name when it is defined. It use keyword “Lambda” to define the function rather than “def”. It can be applied into a variable, pandas apply, and many more.
Use the format `lambda arg1, arg2 : expression` to define the function.

```
#Define a function
penjumlahan = lambda x1, x2: x1 + x2

#Calling the function
penjumlahan(2,4) #Output: 6
```

Introduction to

Basic Module & Packages

The **Module** is a simple Python file that contains collections of functions and global variables and with having a *.py* extension file. It is an executable file and to organize all the modules we have the concept called **Package** in Python.

There are several advantages to modularizing code in a large application:

- Simplicity
- Maintainability
- Reusability
- Scoping

The module or package can be accessed by using **Import** statement.

Basic Module & Packages

How to Make Module

For example, suppose you have created a file called **mod.py** containing the following:

```
s = "Hacktiv8-PTP Python For Data Science"

a = [100, 200, 300]

def foo(arg):
  print(f'arg = {arg}')

class Foo:
  pass
```

Basic Module & Packages

How to Make Module

Assuming **mod.py** is in an appropriate location, which you will learn more about shortly, these objects can be accessed by importing the module as follows:

```
import mod

print(mod.s)
print(mod.a)
mod.foo(['quux', 'corge', 'grault'])
x = mod.Foo()
print(x)
```

Basic Module & Packages

The Module Search Path

The resulting search path is accessible in the Python variable **sys.path**, which is obtained from a module named **sys**:

```
import sys
sys.path
```

Thus, to ensure your module is found, you need to do one of the following:

- Put **mod.py** in the directory where the input script is located or the current directory, if interactive
- Modify the PYTHONPATH environment variable to contain the directory where **mod.py** is located before starting the interpreter
- Or: Put **mod.py** in one of the directories already contained in the **PYTHONPATH** variable
- Put **mod.py** in one of the installation-dependent directories, which you may or may not have write-access to, depending on the OS

Basic Module & Packages

The Import Statement

Module contents are made available to the caller with the **import** statement. To be accessed in the local context, names of objects defined in the module must be prefixed by **mod** :

```
import mod
print(mod.s)
```

An alternate form of the Import statement allows individual objects from the module to be imported directly into the caller's symbol table:

```
from mod import s, foo

print(s)
print(foo('quux'))
```

Basic Module & Packages

The dir() Function

The **dir()** function can be useful for identifying what exactly has been added to the namespace by an **import** statement:

```
>>> from mod import a, Foo
>>> dir()

['Foo', '__annotations__', '__builtins__', '__doc__',
'__loader__', '__name__','__package__', '__spec__', 'a', 'mod']
```

You can also import an entire module under an alternate name:

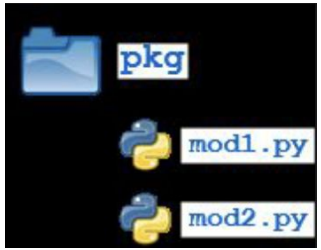
```
>>> import mod
>>> dir(mod)

['Foo', '__builtins__', '__cached__', '__doc__', '__file__',
'__loader__','__name__', '__package__', '__spec__', 'a', 'foo', 's']
```

Basic Module & Packages

Python Packages

Packages allow for a hierarchical structuring of the module namespace using dot notation. In the same way that modules help avoid collisions between global variable names, packages help avoid collisions between module names.



Creating a package is quite straightforward, since it makes use of the operating system's inherent hierarchical file structure.

Here, there is a directory named pkg that contains two modules, **mod1.py** and **mod2.py**. The contents of the modules are:

```
#mod1.py
def foo():
  print('[mod1] foo()')

class Foo:
  pass
```

```
#mod2.py
def bar():
 print('[mod2] bar()')

class Bar:
  pass
```

Basic Module & Packages

Python Packages

With the previous structure, if the **pkg** directory resides in a location where it can be found (in one of the directories contained in **sys.path**), you can refer to the two modules with dot notation (**pkg.mod1**, **pkg.mod2**) and import them with the syntax you are already familiar with:

```
>>> import pkg.mod1, pkg.mod2
>>> pkg.mod1.foo()
[mod1] foo()

>>> from pkg.mod1 import foo
>>> foo()
[mod1] foo()
```

Introduction to

PIP

`Pip` is a package manager for Python. That means it's a tool that allows you to install and manage additional libraries and dependencies that are not distributed as part of the standard library.

You can learn about `pip` supported commands by running it with help:

```
pip help
```

As you can see, `pip` provides an install command to install packages. You can run it to install the requests package:

```
pip install requests
```

You can use the list command to see the packages installed in your environment:

```
pip list
```

HACKTIV8

External References

Colab Link _____ *Visit Here*