Dasar Pemrograman

Python Functions

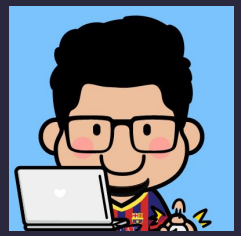
lpdp
lembaga pengelola dana pendidikan

PRAKTISI MENGAJAR

VOKASI KUAT, MENGUATKAN INDONESIA

Kampus Merdeka
INDONESIA JAYA

>>> Hello World



🧑 Hi, I’m lintangwisesa!

💼 Backend Developer @ Smartfren

🎓 S1 Fisika UGM & S2 MTI Fasilkom UI

📧 lintangwisesa@ymail.com

📞 +6285890090045

▶️ Lintang Wisesa

>>> Important Links

Slides, Codes & Tasks:

https://gitlab.com/lintangwisesa26/nusaputra-pm2024

>>> Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

```
# define the function
def my_function():
    print("Hello from a function")

# call & execute the function
my_function()
```

>>> Arguments (args)

Information can be passed into functions as arguments. Arguments are specified after the function name, inside the parentheses. You can add as many arguments, just separate them with a comma.

```
# define a function with argument
def my_function(username):
    print("Hi " + username)

# call & execute the function with argument
my_function(“Lintang”)
```

>>> Built-in Functions

Python has a number of built-in functions. Their documentations are listed on https://docs.python.org/3/library/functions.html. Here are some of them:

```
abs()     # Returns the absolute value of a number
float()   # Returns a floating point number
int()     # Returns an integer number
max()     # Returns the largest item in an iterable
min()     # Returns the smallest item in an iterable
```

>>> Multiple Arguments (args)

```
# define a function with multiple arguments
def my_function(username, lastname):
    printf("Hi {username} {lastname})

# call & execute the function with its arguments
my_function(“Lintang”, “Wisesa”)
```

>>> Default Arguments Value

```
# define a function with default argument value
def my_function(username="Andy"):
    printf("Hi {username})

# call & execute the function
my_function("Lintang")
my_function()
```

>>> Arbitrary Arguments (*args)

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition. This way the function will receive a tuple of arguments, and can access the items accordingly.

```
# define a function with arbitrary argument
def my_function(*name):
    printf("Hi {name[1]} {name[2]}")

# call & execute the function with its *args
my_function("Lintang", "Wisesa")
```

>>> Keyword Arguments (kwargs)

You can also send arguments with the key = value syntax. This way the order of the arguments does not matter.

```
# define a function with keyword argument
def my_function(a, b, c):
    print(a + b + c)

# call & execute the function with its kwargs
my_function(a = 2, b = 4, c = 6)
my_function(c = 10, b = 5, a = 1)
```

>>> Arbitrary Keyword Arguments (**kwargs)

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition. This way the function will receive a dictionary of arguments, and can access the items accordingly.

```
# define a function with arbitrary keyword argument
def my_function(**data):
    printf("Data: {data['age']} {data['job']}")

# call & execute the function with its **kwargs
my_function(age = 20, job = "Student")
```

>>> Return Functions

To let a function return a value, use the return statement:

```
def my_function(x):
    return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

>>> Lambda Functions

A lambda function is **a small anonymous function**. A lambda function can take any number of arguments, but can only have one expression.

```
x = lambda a : a + 10
print(x(2))
print(x(90))
```

>>> Recursive Function

Recursion is a common mathematical and programming concept. It means that **a function calls itself**.

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