

## Python Functions

Review the topic of functions in Python with this e-book. Check out the table of contents to navigate to each topic.

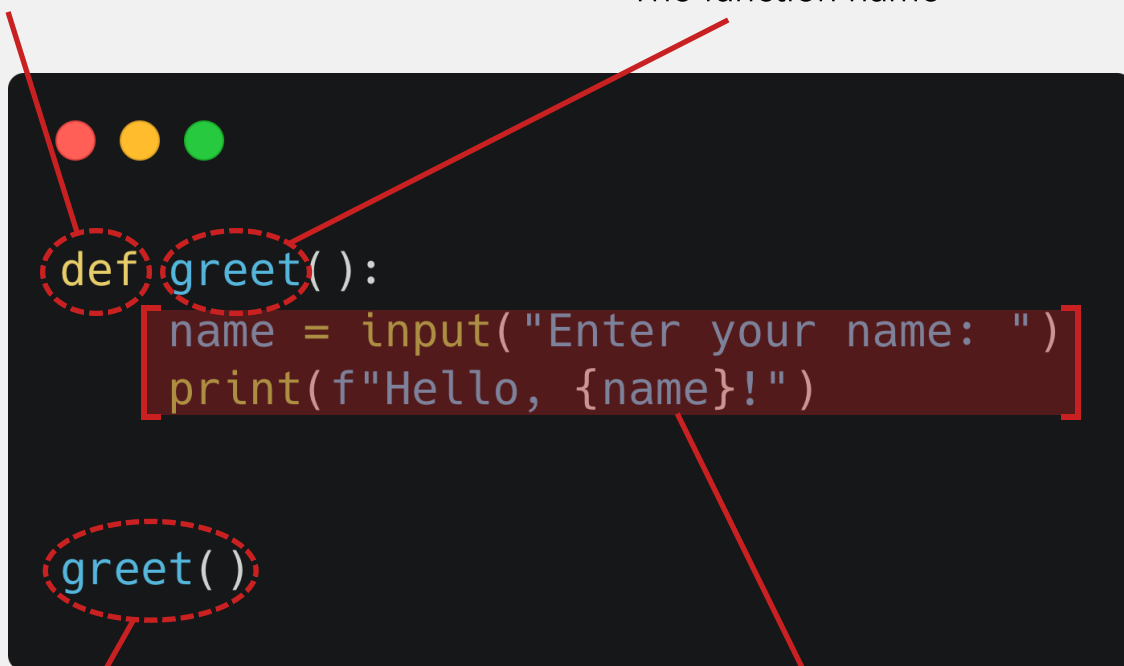
### Contents

- 1: Python Functions
- 2: The 'def' and 'return' keywords
- 3: Indentation
- 4: Parameters and Arguments
- 6: Lambda Expressions

## Python Functions

Used to start a function definition

The function name



Calls the function, which jumps to the function body

Function body, runs when we execute the function

Must be indented more than the def line

## The `return` keyword

By default, functions return `None`

`pass` is a keyword which means "do nothing".

It's necessary since a function needs a body.

```
def greet():  
    pass # Do nothing  
  
# print the function's return  
print(greet()) # None
```

You can control what a function returns using the `return` keyword.

A function can return any value.

```
def greet():  
    name = "John Doe"  
    return f"Hello, {name}!"  
  
# printing the function  
print(greet()) # John Doe
```

## Parameters

A parameter is the variable defined inside the function's definition parentheses.

```
def greet(name):  
    return f"Hello, {name}!"
```

A function can have any number of parameters

### Extra Resources

1. [What are functions in Python?](#)
2. [Defining our first Python function](#)
3. [Getting values out of a function](#)

## Arguments

When a function is called, an argument is the value that is passed to it

To call a function that has a parameter defined, you have to provide an argument

```
def greet(name):  
    return f"Hello, {name}!"  
  
print(greet("John Doe")) # Hello, John Doe!
```

## Flow of data in functions

When a function is called, arguments are assigned to parameters (1). The parameters can be used within the function (2). Return values are given back to the caller (3).

```
def greet(name):  
    return f"Hello, {name}"  
  
greeting = greet("Rolf Smith")  
print(greeting) # Hello, Rolf Smith
```

### Extra Resources

1. [Function parameters and arguments](#)

## Named Arguments

If you want to be specific about which arguments you pass to functions, you can use named arguments. This also allows you to pass arguments in any order you wish.

```
def greet(name, age):  
    return f"{name} is {age} years old!"  
  
print(greet(name="John Doe", age=31))
```

## Default Parameter Values


Parameters can have default values. When passing an argument to the function, the default parameter value will be overwritten by the given argument:

```
def greet(name="John Doe"):  
    return f"Hello, {name}!"  
  
print(greet())  
# Hello, John Doe!
```


```
def greet(name="John Doe"):  
    return f"Hello, {name}!"  
  
print(greet("Jimmy boy"))  
# Hello, Jimmy boy!
```

If the function has a mix of parameters with and without default parameters, the default parameters must be defined last

```
def greet(age, name="John Doe"):  
    return f"{name} is {age} years old!"  
  
print(greet(32))
```



```
def greet(name="John Doe", age):  
    return f"{name} is {age} years old!"  
  
print(greet(32))
```



# Python Lambda Expressions

Lambda expressions are an alternative syntax for defining simple functions.

```
lambda name: f"Hello, {name}"
```

This function has one parameter, `name`, and returns `f"Hello, {name}"`.

Lambda functions:

- Don't have a name
- Can be assigned to a variable
- Can be difficult to read

Because of this, they are used only in very specific cases.

Calling a lambda inline requires multiple pairs of brackets:

```
result = (lambda x, y: x + y)(15, 3)
```

Lambda function

2 arguments

Returns the sum of its 2 arguments

Arguments: x=15, y=3

## Extra Resources

1. [Lambda Expressions](#)

# First Class Functions

Functions in Python are "first class citizens".

They can be stored in data structures, passed as arguments, and stored in variables.

```
def avg(*args):  
    return sum(args) / len(args)
```

avg function stored in the dictionary

```
operations = {  
    "average": avg,  
    "total": sum,  
    "top": max  
}
```

Retrieves the avg function as a value

```
selection = "average"  
operation = operations[selection]  
operation(1, 3, 5)
```

Calls the `avg` function

This is akin to doing `avg(1, 3, 5)`