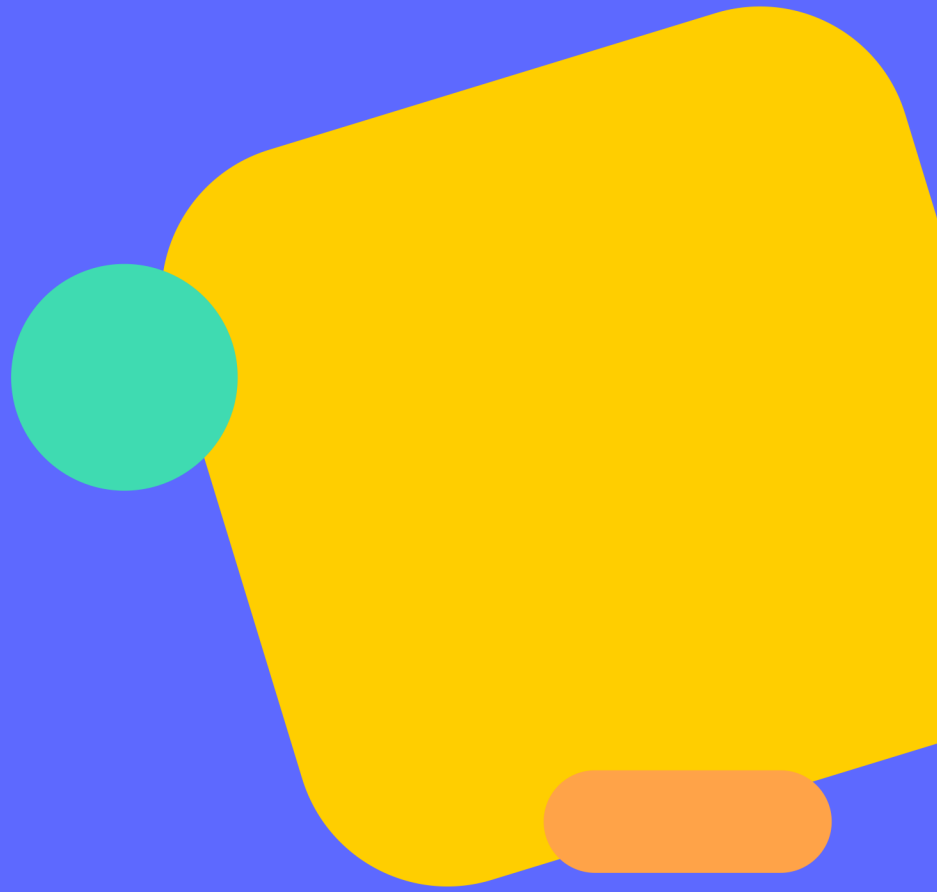
Artificial Intelligence

Python Functions

Powered By

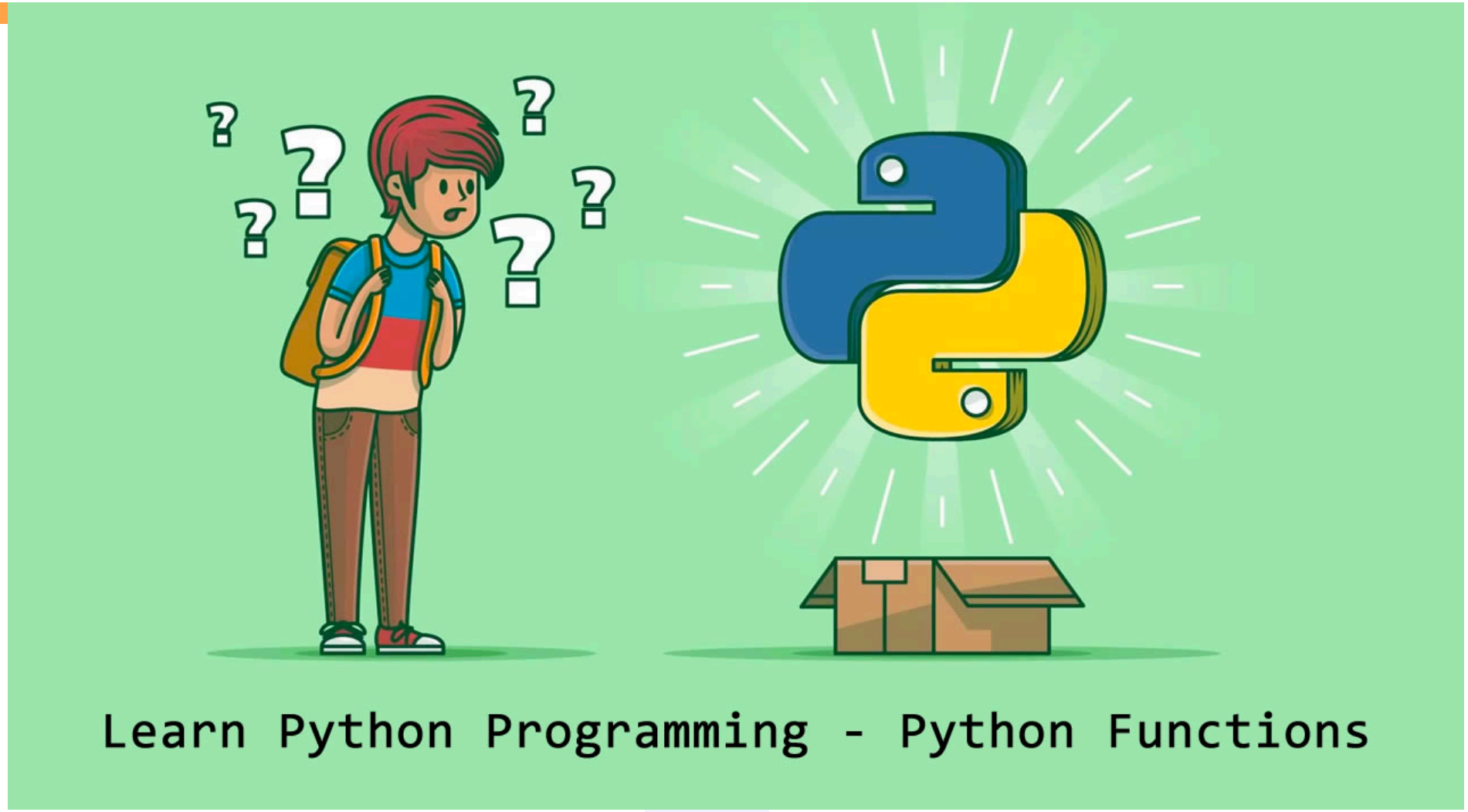
CodingHero



Topic Overview

- ✓ Why to use Functions?
- ✓ Python Functions
- ✓ Python Functions Advantages & Its Types
- ✓ User Defined Functions
- ✓ User Defined Function Arguments
- ✓ Scope and Lifetime of variables
- ✓ Local vs Global Variables

Python Functions



Why to use Functions ?

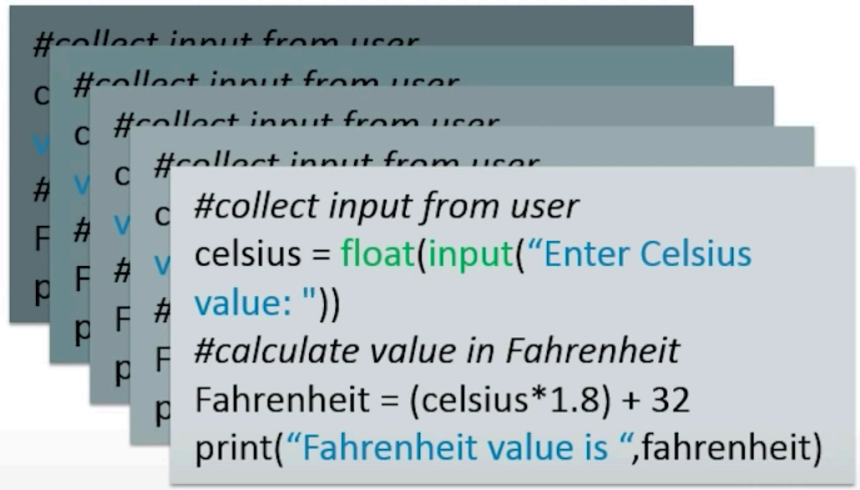
You write a program in which ***Celsius*** must be converted to ***Fahrenheit*** multiple times

```
#collect input from user
celsius = float(input("Enter Celsius value: "))
#calculate value in Fahrenheit
Fahrenheit = (celsius*1.8) + 32
print("Fahrenheit value is ",fahrenheit)
```

Program to calculate *Fahrenheit*

Fahrenheit = (9/5)Celsius + 32

Logic to calculate *Fahrenheit*



You wouldn't want to **repeat** those **same lines of code** every time a value needed conversion

Python Functions

Functions are a **set of related statements grouped together to carry out a specific task**. Functions **divide our code into useful blocks**, make it more **readable**, **allow us to reuse it** and save some time.

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

Python Functions Advantages & Its Types

The advantages of using functions are:

•Helps in **avoiding repetition** i.e functions **reduce duplicity**.
•Decomposing complex problems into simpler pieces. So, functions **make our program more organized and manageable.**
•Improving **clarity** of the code
•Helps in making the **code reusable**.
•Information hiding

There are four types of functions in Python:

Python **Built-in** Functions (already created, or predefined, function)
Python **Recursion** Functions
Python **Lambda** Functions
Python **User-defined** Functions (a function created by users as per the requirements)

User Defined Functions

User defined function:

```
# Defining a function
def fahr_to_celsius(temp):
    FtoC = ((temp-32)*(5/9))
    return FtoC

# Calling a function
fahr_to_celsius(102)
```

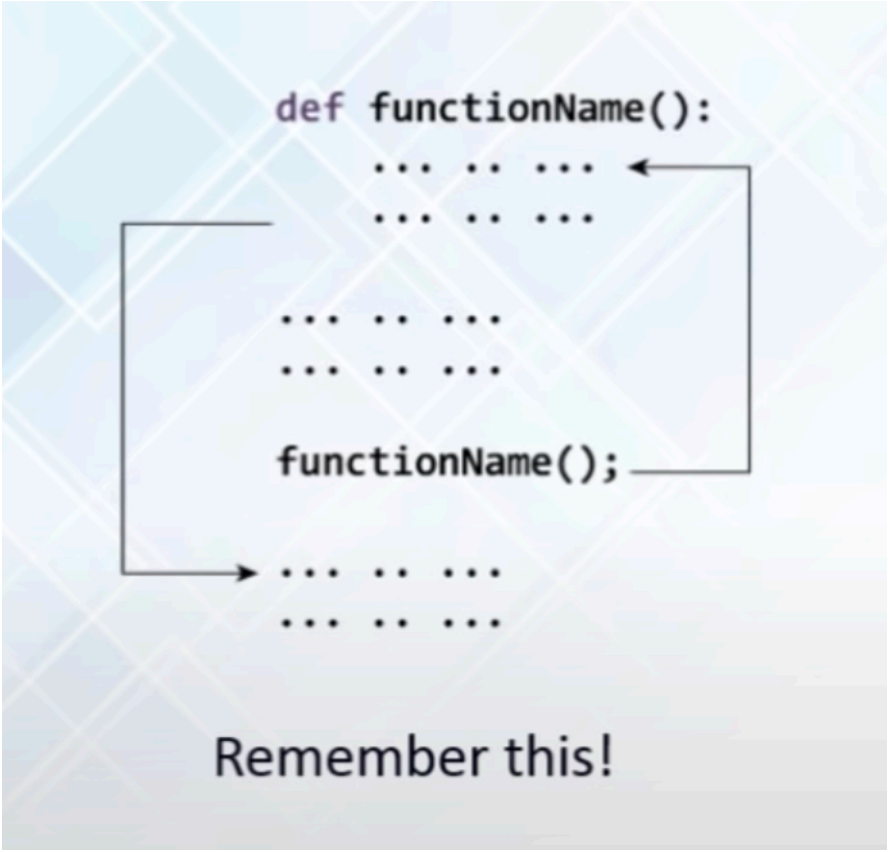
Defining a function

Function body

Return statement

Calling the function

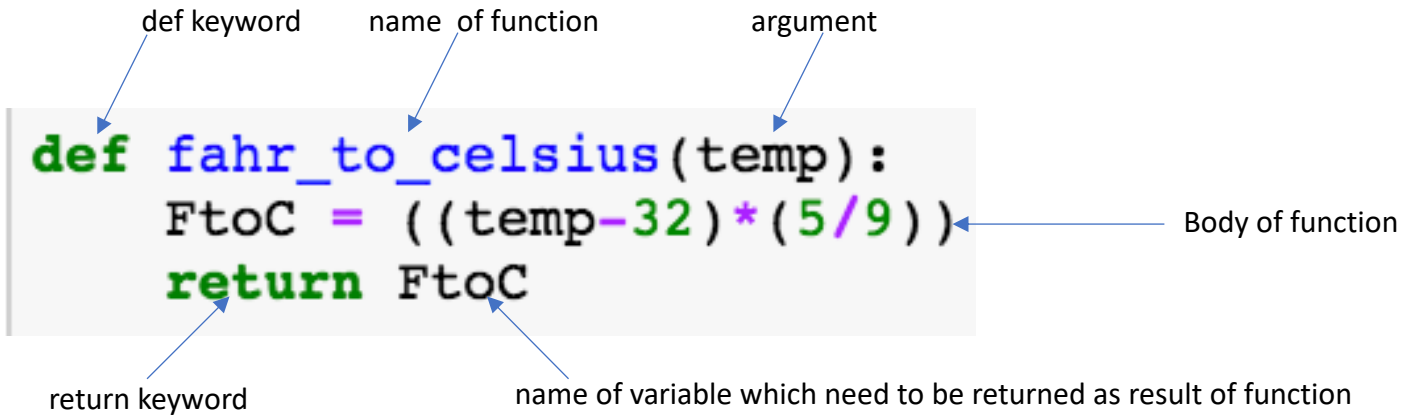
```
38.888888888889
```



User Defined Functions

Defining a Function in Python

- The **def keyword** is used to start the function definition.
- The def keyword is followed by a **function-name** which is followed by **parentheses** containing the **arguments** passed by the user and a **colon** at the end.
- After adding the colon, the **body of the function** starts with an indented block in a new line.
- The **return statement sends a result back**. **A return statement with no argument is equivalent to return none statement.**



User Defined Functions

Defining a Function in Python

```
def fahr_to_celsius(temp):
    return ((temp-32)*(5/9))
```

```
def celsius_to_kelvin(temp_c):
    return temp_c + 273.15
```

```
def fahr_to_kelvin(temp_f):
    temp_c = fahr_to_celsius(temp_f)
    temp_k = celsius_to_kelvin(temp_c)
    return temp_k
```

User Defined Functions

Calling a Function In Python

Defining a function only structures the code blocks and gives the function a name. To execute a function, we have to call it.

```
fahr_to_celsius(102)
```

```
38.88888888888889
```

```
print('freezing point of water:', fahr_to_celsius(32), 'C')
print('boiling point of water:', fahr_to_celsius(212), 'C')
```

```
freezing point of water: 0.0 C
boiling point of water: 100.0 C
```

```
print('freezing point of water in Kelvin:', celsius_to_kelvin(0.))
```

```
freezing point of water in Kelvin: 273.15
```

```
print('boiling point of water in Kelvin:', fahr_to_kelvin(212.0))
```

```
boiling point of water in Kelvin: 373.15
```

User Defined Functions

Docstring:
The first string after the function header is called the docstring and is short for documentation string. It is briefly used to explain what a function does. It is optional.

Return statement:
The return statement is used to exit a function. This statement can contain an expression that gets evaluated and the value is returned. **If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object**.
return [expression_list]

The pass Statement:
If you have a function definition with no content for some reason, put in the pass statement to avoid getting an error

User Defined Function Arguments

Function with Arguments(args):

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

```
def greet(name, msg):
    """This function greets to
    the person with the provided message"""
    print("Hello", name + ', ' + msg)
greet("Monica", "Good morning!")
```

```
Hello Monica, Good morning!
```

From a function's perspective:
A **parameter** is the variable listed inside the parentheses in the function definition.
An **argument** is the value that is sent to the function when it is called.

Function with 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:

```
def greet(*names):
    """This function greets all
    the person in the names tuple."""
    # names is a tuple with arguments
    for name in names:
        print("Hello", name)
greet("Monica", "Luke", "Steve", "John")
```

```
Hello Monica
Hello Luke
Hello Steve
Hello John
```

User Defined Function Arguments

Function with Keyword Arguments(kwargs):

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

```
def greet(name, msg):
    """
    This function greets to
    the person with the
    provided message.

    If the message is not provided,
    it defaults to "Good
    morning!"
    """
    print("Hello", name + ', ' + msg)
# 2 keyword arguments
greet(name = "Katrina",msg = "How do you do?")
# 2 keyword arguments
greet(msg = "How are you?",name = "Bhavna")
#1 positional, 1 keyword argument
greet("John", msg = "Can we meet tomorrow?")
```

```
Hello Katrina, How do you do?
Hello Bhavna, How are you?
Hello John, Can we meet tomorrow?
```

User Defined Function Arguments

Function with 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:

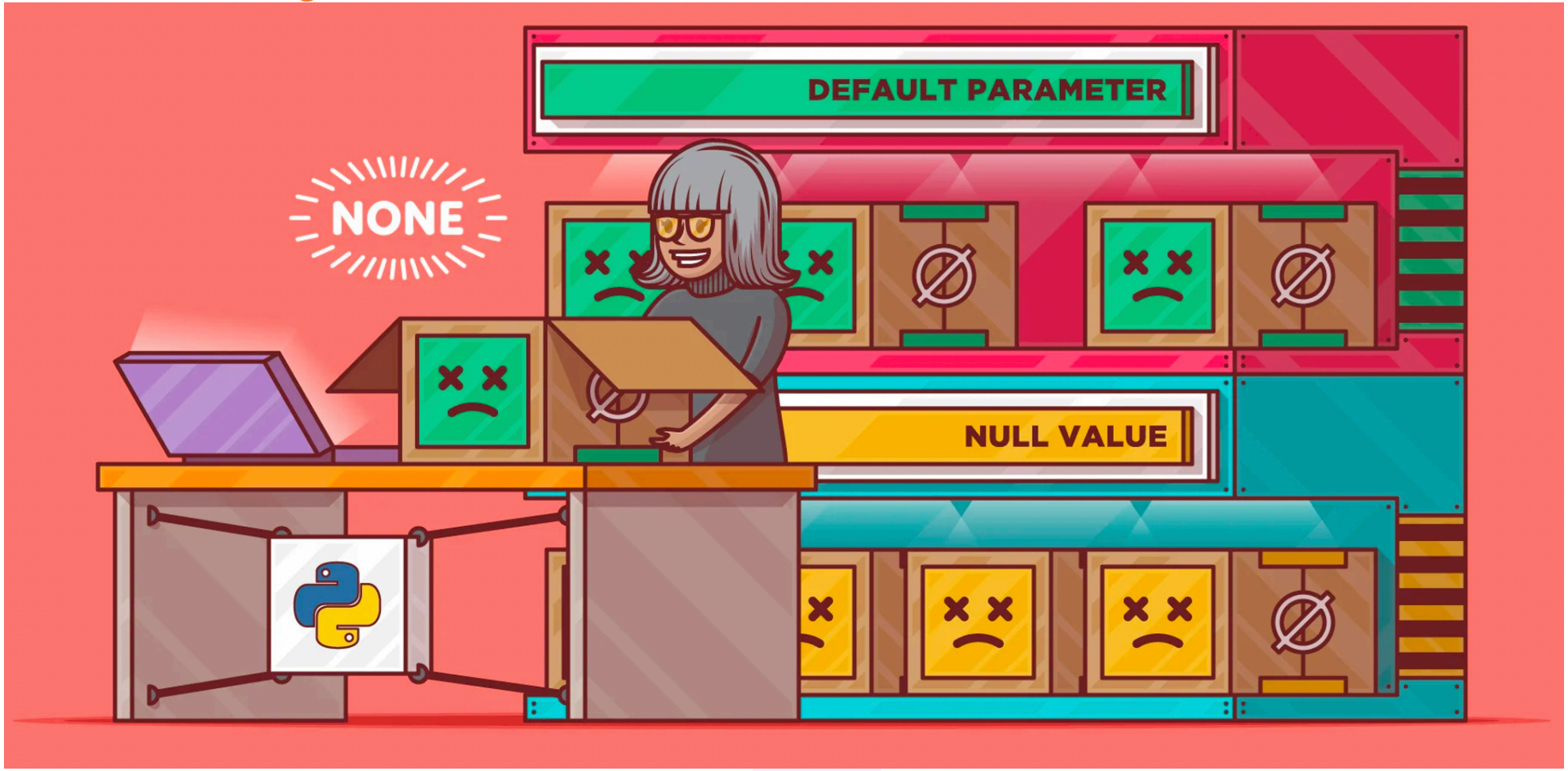
You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.
E.g. if you send a List as an argument, it will still be a List when it reaches the function:

```
def greet(**kwargs):
    for key,value in kwargs.items():
        print("Hello!",key,',',value)
greet(Monika="How are you?",chitra="How old are you?")
```

```
Hello! Monika , How are you?
Hello! chitra , How old are you?
```

User Defined Function Arguments

Function with Default Arguments:



User Defined Function Arguments

Function with Default Arguments:

Default arguments are those arguments that take default values if no explicit values are passed to these arguments from the function call. Let's define a function with one default argument.

It's important to note that **parameters with default arguments cannot be followed by parameters with non default argument**. Take the following function for example:

```
def find_square(integer1=2):
    result = integer1 * integer1
    return result
```

```
find_square()
```

```
4
```

```
find_square(10)
```

```
100
```

```
def take_power(integer1=2, integer2):
    result = 1
    for i in range(integer2):
        result = result * integer1

    return result
```

```
  File "<ipython-input-45-d745edd8e5c1>", line 1
    def take_power(integer1=2, integer2):
                   ^
SyntaxError: non-default argument follows default argument
```

Scope and Lifetime of variables:

Scope and Lifetime of variables:

The **scope** of a **variable** is the part of the program within which the **variable** can be used .Their availability depends on where they are defined.

Similarly, The **lifetime of a variable** is the time duration for which memory is allocated to store it,

Depending on the scope and the lifetime, there are two kinds of variables in Python.

Local Variables:
Global Variables:

Local Variables vs Global Variables

Local Variables vs Global Variables

- **Variables or parameters defined inside a function** are called local variables as their scope is limited to the function only. On the contrary, Global variables are **defined outside of the function**.

- Local variables **can't be used outside the function** whereas a global variable **can be used throughout the program anywhere** as per requirement.

- The **lifetime** of a local variable **ends with the termination or the execution of a function**, whereas the lifetime of a global variable **ends with the termination of the entire program.**

- The variable defined inside a function **can also be made global by using the global statement.**

Here is an example to illustrate the scope of a variable inside a function.

```
def my_func():
    x = 10
    print("Value inside function:",x)

x = 20
my_func()
print("Value outside function:",x)
```

```
Value inside function: 10
Value outside function: 20
```

Local Variables vs Global Variables

Local Variable Example

```
def func3():
    a=7
    print(a)
```

```
func3()
print(a)
```

```
7

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-53-4130de61ea31> in <module>
      1 func3()
----> 2 print(a)

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

However, you can't change its value from inside a local scope(here, inside a function). To do so, you must declare it global inside the function, using the 'global' keyword.

Global Variable Example

```
def func4():
    global y
    y=9
    print(y)
func4()
print(y)
```

```
9
9
```

References:

https://problemsolvingwithpython.com/04-Data-Types-and-Variables/04.02-Boolean-Data-Type/

https://www.codesdope.com/practice/python-boolean/

https://pynative.com/python-operators-and-expression-quiz/

https://realpython.com/quizzes/python-operators-expressions

https://www.softwaretestinghelp.com/python/python-conditional-statements/