Python Functions

FPT Education
FPT UNIVERSITY

Function Declaration and Arguments

Global and Local Variables

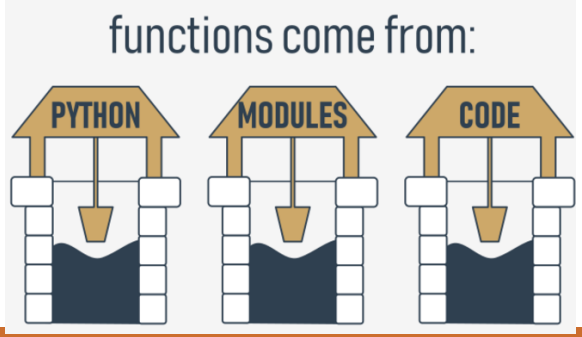
Anonymous Function

Problem Solving: Customer Segmentation

Why do we need functions?

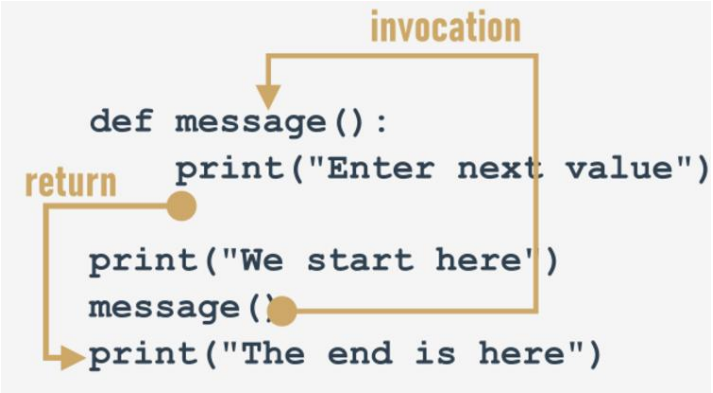
- if a particular fragment of the code begins to appear in more than one place, consider the possibility of isolating it in the form of a functions in your program.
- if a piece of code becomes so large that reading and understating it may cause a problem, consider dividing it into separate, smaller problems, and implement each of them in the form of a separate function.
- decompose the problem to allow the product to be implemented as a set of separately written functions packed together in different modules.

- **Where do the functions come from?**



How functions work

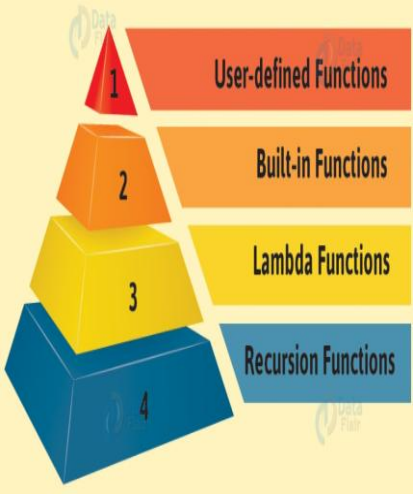
- It tries to show you the whole process:
  - when you invoke a function, Python remembers the place where it happened and jumps into the invoked function;
  - the body of the function is then executed;
  - reaching the end of the function forces Python to return to the place directly after the point of invocation.
- You mustn't invoke a function which is not known at the moment of invocation.




```
print("We start here.")
message()
print("We end here.")


def message():
    print("Enter a value: ")
```

Function Declaration and Arguments

- A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.
  - built-in functions which are an integral part of Python (such as the print() function). the ones that come from pre-installed modules
  - user-defined functions which are written by users for users - you can write your own functions and use them freely in your code,
  - the lambda functions



Function Declaration and Arguments

- *As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.*
- *Example:*

```
def hello():
    """
    This Python function simply prints
  hello to the screen
    """
    print("Hello")
```

Function Declaration and Arguments

- *It is recommended to understand what are arguments and parameters before proceeding further. Vocabulary parameters and arguments are not limited to python but they are same across different programming languages.*
  - ***Arguments*** *are values that are passed into function(or method) when the calling function*
  - ***Parameters*** *are variables(identifiers) specified in the (header of) function definition*
  - *Example :*

```
# Function Definition
def add(a, b):
    return a + b
```

Parameters

```
# Function Call
add(2, 3)
```

Arguments

Function Declaration and Arguments

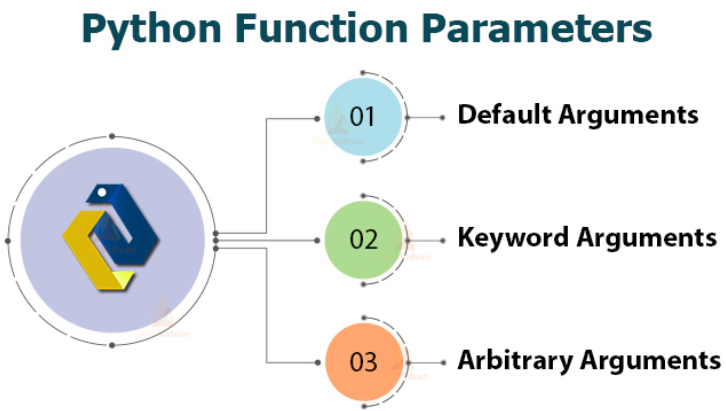
FPT Education
FPT UNIVERSITY

Function Declaration

```
def add(x, y):
    print(f'arguments are {x} and {y}')
    return x + y
```

1. def keyword
2. function name
3. function arguments inside ()
4. colon ends the function definition
5. function code
6. function return statement

Function Arguments



Function Declaration and Arguments

FPT Education
FPT UNIVERSITY

Positional arguments

Parameter 1 Parameter 2

```
def love_you(my_name, your_name):
    print(f"{my_name} loves {your_name}"
```

```
love_you("정우성","아이유")
```

"정우성 loves 아이유"

Keyword arguments

Parameter 1 Parameter 2

```
def love_you(my_name, your_name):
    print(f"{my_name} loves {your_name}"
```

```
love_you(your_name = "아이유", my_name = "정우성",)
```

"정우성 loves 아이유"

Function Declaration and Arguments

FPT Education
FPT UNIVERSITY

Mixing positional and keyword arguments

Parameter 1 Parameter 2

```
def love_you(my_name, your_name):
    print(f"{my_name} loves {your_name}"
```

```
love_you("정우성", your_name = "아이유")
```

"정우성 loves 아이유"

Default value parameters

Parameter 1 Parameter 2

```
def love_you(my_name, your_name = "아이유"):
    print(f"{my_name} loves {your_name}"
```

```
love_you("정우성")
```

"정우성 loves 아이유"

함수를 호출 할때
2번째 parameter 값을
정의하지 않았으나 default 값이
정의 되어 있음으로 default 값이 자동으로 사용된다.

Parameterized functions

FPT Education
FPT UNIVERSITY

- A parameter is actually a variable, but there are two important factors that make parameters different and special:
  - parameters exist only inside functions in which they have been defined, and the only place where the parameter can be defined is a space between a pair of parentheses in the def statement;
  - assigning a value to the parameter is done at the time of the function's invocation, by specifying the corresponding argument.
  - parameters live inside functions (this is their natural environment)
  - arguments exist outside functions, and are carriers of values passed to corresponding parameters.
- A function can have as many parameters as you want, but the more parameters you have, the harder it is to memorize their roles and purposes.

Parametrized functions

- It happens at times that a particular parameter's values are in use more often than others. Such arguments may have their default (predefined) values taken into consideration when their corresponding arguments have been omitted.

```
def introduction(first_name, last_name="Smith"):
    print("Hello, my name is", first_name, last_name)
```

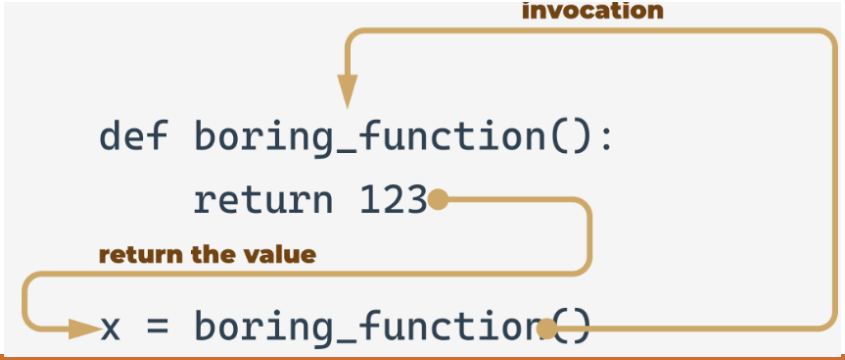
```
introduction("James", "Doe")

introduction("Henry")
```

```
def add_numbers(a, b=2, c):
    print(a + b + c)

add_numbers(a=1, c=3)
```

Returning a result from a function

- return without an expression: it causes the immediate termination of the function's execution, and an instant return (hence the name) to the point of invocation.
- return with an expression
  - it causes the immediate termination of the function's execution (nothing new compared to the first variant)
  - moreover, the function will evaluate the expression's value and will return (hence the name once again) it as the function's result.

```
def boring_function():
    print("'Boredom Mode' ON.")
    return 123

print("This lesson is interesting!")
boring_function()
print("This lesson is boring...")
```



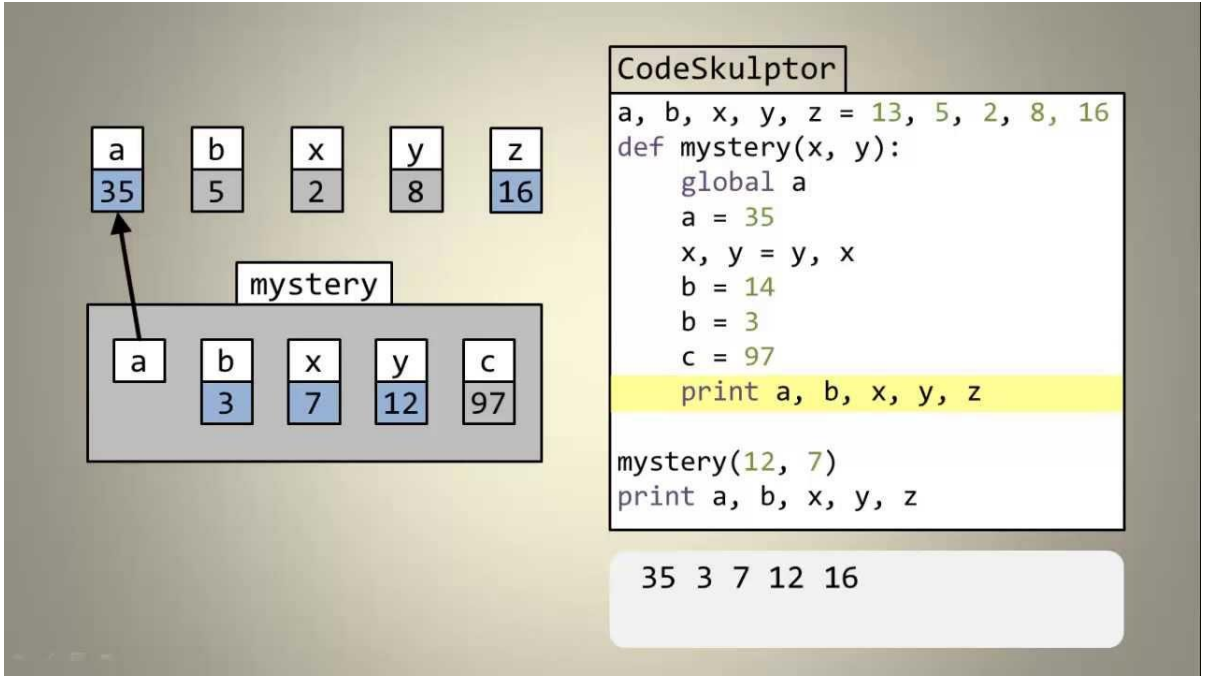
Global and Local Variables

FPT Education
FPT UNIVERSITY

- *Python global variable is defined outside a function. It will work on the whole function if there is no the same named variable.*
- *Variables defined within a function are local variables.*
- In the picture next slide, the scope of the local variable is highlighted blue – it's the function where that var was defined.
- Any code inside that function can access (read and change) this variable. Any code outside it can't. It's local, so it's invisible from outside.

Global and Local Variables

- *Example*



Global and Local Variables

- *Python namespaces can be divided into four types (cont.)*
  - ***Local Namespace**: A function, for-loop, try-except block are some examples of a local namespace. The local namespace is deleted when the function or the code block finishes its execution.*
  - ***Enclosed Namespace**: When a function is defined inside a function, it creates an enclosed namespace. Its lifecycle is the same as the local namespace.*

Global and Local Variables

FPT Education
FPT UNIVERSITY

- *Python namespaces can be divided into four types*
  - ***Global Namespace**: it belongs to the python script or the current module. The global namespace for a module is created when the module definition is read. Generally, module namespaces also last until the interpreter quits.*
  - ***Built-in Namespace**: The built-in namespace is created when the Python interpreter starts up and it's never deleted.*
- *Scope defines the hierarchy in which we search for a variable*

Global and Local Variables

- A variable that exists outside a function has a scope inside the function body (Example 1) unless the function defines a variable of the same name (Example 2, and Example 3)
- A variable that exists inside a function has a scope inside the function body (Example 4)

```
def mult(x):
    var = 7
    return x * var


var = 3
print(mult(7))     # outputs: 49
```

```
var = 2
print(var)     # outputs: 2


def return_var():
    global var
    var = 5
    return var


print(return_var())     # outputs: 5
print(var)     # outputs: 5
```

Global and Local Variables

FPT Education
FPT UNIVERSITY

Namespaces

```
x = 10
y = 20

def outer():
    z = 30

    def inner():
        x = 30
        print(f'x is {x}')
        print(f'z is {z}')
        print(f'y is {y}')
        print(len("abc"))

    inner()

outer()
```

global

enclosed

local

local

built-in namespace has len() function

Error Case

```
x = "Global Scope"

def function():
    y = "Local Scope"
    print(y)

print(y)
```

Output

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

Global and Local Variables

FPT Education
FPT UNIVERSITY

Enclosing Scope

Built-in Scope print()

```
x = "Global Scope"

def outer_func():
    x = "Enclosing Scope"

    def inner_func():
        x = "Local Scope"
        print(x)

    inner_func()

outer_func()
```

Explanation

If we are talking about the function `outer_func`, then:

- `y` belongs to Local Scope
- `x` belongs to Global Scope

If we are talking about the function `inner_func`, then:

- `z` belongs to Local Scope
- `y` belongs to Enclosing Scope
- `x` belongs to Global Scope

Global and Local Variables

FPT Education
FPT UNIVERSITY

Case 1

```
x = "Global Scope"

def outer_func():
    x = "Enclosing Scope"

    def inner_func():
        x = "Local Scope"
        print("I found x in", x)

    inner_func()

outer_func()
```

Output

Case 2

```
x = "Global Scope"

def outer_func():
    x = "Enclosing Scope"

    def inner_func():
        print("I found x in", x)

    inner_func()

outer_func()
```

Output

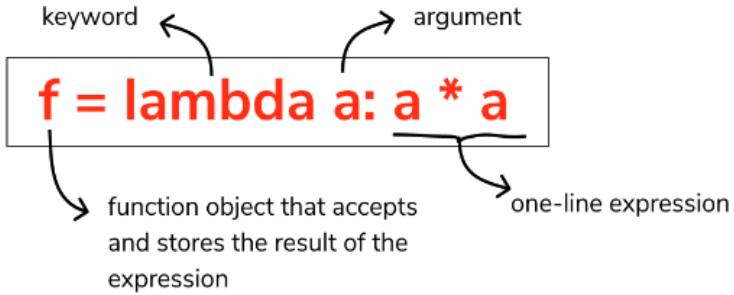
Case 3

```
x = "Global Scope"

def outer_func():

    def inner_func():
        print("I found x in", x)

    inner_func()

outer_func()
```

Output

Anonymous Function

- *An anonymous function is a function that is defined without a name. While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword.*
- *These functions replace traditional user-defined functions having two or more lines with a simple one-line function. Creating the Lambda function is faster than creating it using the traditional way. It can have multiple arguments with one expression*
- *Example*



Anonymous Function

Example 1: *def square(x):*

*return x ** 2*

- *square_lambda = lambda x: x ** 2*
- *print(square(10))*
- *print(square_lambda(10)) ==> Output : 100 100*

Example 2:

- *rectangle_area = lambda x, y: x * y*
- *print(f'Area of Rectangle (4, 5) is {rectangle_area(4, 5)}')*
- *Output : Area of Rectangle (4, 5) is 20*

Anonymous Function

- *The map() function takes a function and an iterable as the arguments. The function is applied to every element in the iterable and the updated iterable is returned.*

Example:

```
numbers = [10, 45, 23, 56, 6, 34, 78, 90, 3]
new_numbers = list(map(lambda x: x + 2, numbers))
print(new_numbers)
```

Output: [12, 47, 25, 58, 8, 36, 80, 92, 5]

Anonymous Function

- *The built-in filter() function takes a function and an iterable as the argument. The function is applied to each element of the iterable. If the function returns True, the element is added to the returned iterable.*

Example:

```
numbers = [10, 45, 23, 56, 6, 34, 78, 90, 3]
filtered_list = list(filter(lambda x: x > 50, numbers))
print(filtered_list)
Output : [56, 78, 90]
```

Anonymous Function

- *The reduce() function is present in the functools module. This function takes a function and a sequence as the argument. The function should accept two arguments. The elements from the sequence are passed to the function along with the cumulative value. The final result is a single value.*

Example:

```
from functools import reduce
list_ints = [1, 2, 3, 4, 5, 6]
total = reduce(lambda x, y: x + y, list_ints)
print(f'Sum of list_ints elements is {total}')
```

Output : 21

Anonymous Function

FPT Education
FPT UNIVERSITY

Comparison

Traditional Functions	Lambda Functions
They can have multiple expressions/ statements in their body.	They can have just a single expression in their body.
These can have default values for their parameters.	Lambda parameters can't have default values.
Normal function can return an object of any type.	Lambda function always returns a function object.
It takes more time for execution as compared to lambda functions.	These take relatively less time.
Require more lines of code.	It requires a maximum of two lines of code to define and call a lambda function.

Problem Solving

Customer Segmentation :

- Segmentation is the process of dividing your customers up into different groups, with each group sharing similar characteristics, to improve engagement, sales and loyalty.

Problem Solving

- Customer Segmentation :