எங்கள் வாழ்வும் எங்கள் வளமும்
மங்காத தமிழ் என்று சங்கே முழங்கு ... *புரட்சிக்கவி*

NOTICE

➢ We support open-source products to spread Technology to the mass.

➢ This is completely a FREE training course to provide introduction to Python language

➢ All materials / contents / images/ examples and logo used in this document are owned by the respective companies / websites. We use those contents for FREE teaching purposes only.

Thanks to all the open-source community and to the below websites from where we take references / content /code example. definitions, please use these websites for further reading:

- Python Notes For Professionals.pdf – this is the book we follow
- https://docs.python.org
- https://www.datacamp.com/community/tutorials/functions-python-tutorial
- https://www.w3schools.com
- https://data-flair.training/blogs/python-function/
- https://problemsolvingwithpython.com/07-Functions-and-Modules/07.07-Positional-and-Keyword-Arguments/
- https://www.tutorialsteacher.com/python/callable-method

Today's class

Data Science in Tamil

007 TOPIC: PYTHON FUNCTIONS

- What is function
- what is function signature in python?
- Rules for naming python function (identifier)
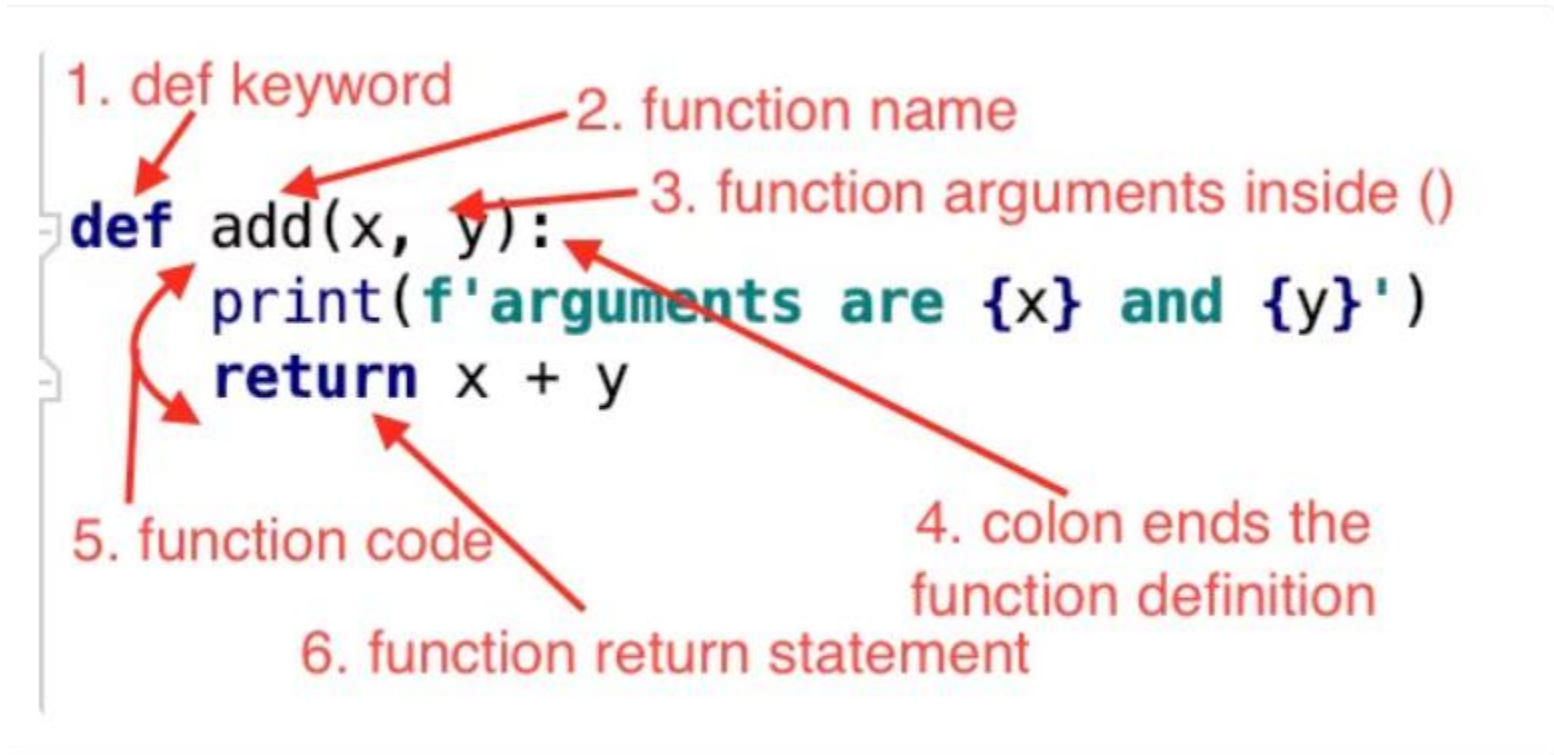- The pass Statement in a function

Types of Functions in Python

1. Function with no parameter and with No Return value.

2. Function with <mark>parameter and No Return value.</mark>

3. Function with <mark>parameter</mark> and return value.

4. Returning Multiple Values in a function

5. Positional arguments

6. Function with default arguments. – Default arguments are optional arguments

7. Function with arbitrary positional arguments.

8. Built in Functions

9. User defined functions

10.   Function with arbitrary keyword arguments.

   – *kargs

   – **kwargs

11. High Order Functions

12.   Anonyms Functions / Lambda Functions

13.   Recursion functions

<mark>PS: Callable fn (dict)</mark>

What is function

```
1. def keyword          2. function name
                         3. function arguments inside ()
def add(x, y):
    print(f'arguments are {x} and {y}')
    return x + y

5. function code         4. colon ends the
                            function definition
6. function return statement
```

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

Python function in any programming language is a sequence of statements in a certain order, given a name. When called, those statements are executed. So we don't have to write the code again and again for each [type of] data that we want to apply it to. This is called code re-usability

function is a piece of code written to carry out a specified task. To carry out that specific task, the function might or might not need multiple inputs.

- A function is a block of code with a name.
- We can call a function by its name.
- The code inside a function only runs when it's called.
- A function can accept data from the caller program, it's called as function parameters.
- The function parameters are inside parentheses and separated by a comma. A function can accept any number of arguments.
- A function can return data to the caller program. Unlike other popular programming languages, Python functions definition doesn't specify the return type.
- We can't use reserved keywords as the function name. A function name must follow the Python identifiers definition rules.

Rules to follow to naming python function

1. Same rules of declaring variables
2. It can begin with either of the following: A-Z, a-z, and underscore(_).
3. The rest of it can contain either of the following: A-Z, a-z, digits(0-9), and underscore(_).
4. A reserved keyword may not be chosen as an identifier.

What is function signature in python?

1. parameters and their types
2. it returns a value and type, return can returns MULTIPLE VALUES

1- Function with no argument and with a No Return value.
Defining and call a function

```python
def my_function():
  print("Say Hello from a function")

my_function()
```

2- Python Function with parameter and No Return value.

➢ Information can be passed into functions as arguments.
➢ 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.

> 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.

```python
def my_function(mySubject):
  print("I am studying ", mySubject)

my_function("Python")
my_function("Numpy")
my_function("Pandas")
```

output

I am studying Python

I am studying Numpy

I am studying Pandas

3 - Function with argument and return value.

```python
def add_Fucntion(a, b):
    result = a + b
    return result

add_Fucntion(2, 2)
```

```python
def add_Function(internalMark, externaMark):
    totalMarkPlus = internalMark + externaMark
    totalMarkSub = internalMark - externaMark
    totalMarkMulti = internalMark * externaMark
    return [totalMarkPlus, totalMarkSub, totalMarkMulti]

a= add_Function(5,25)
print(a)
print(type(a))

print("----------------")

a, b, c = add_Function(5,25)
print(a)
print(b)
print(c)
print(type(a))
print(type(b))
```

----

4 - Returning Multiple Values in a function via return statement

```
def add_Fucntion(a, b):
    addResult = a + b
    subResult = a * b
    multiResult = a * b

    return addResult,subResult,multiResult

result = add_Fucntion(10,2)
print(result)
```

5-Positional arguments

An <mark>argument</mark> is a **variable, value, object, or function** passed to a function or method as input. Positional arguments are arguments that need to be included in the proper position or order.

The first positional argument always needs to be listed first when the function is called. The second positional argument needs to be listed second and the third positional argument listed third, etc.

```python
def add_Function(internalMark, externaMark):
    totalMarkPlus = internalMark + externaMark
    print("My Internal mark is ", internalMark)
    print("My External mark is ", externaMark)
    return totalMarkPlus

result =  add_Function(externaMark=75, internalMark=25)
print(result)
```

6-Function with default arguments. - Default arguments are optional arguments

```python
def add_Function(externaMark, internalMark=51):
    totalMarkPlus = internalMark + externaMark
    print("My Internal mark is ", internalMark)
    print("My External mark is ", externaMark)
    return totalMarkPlus

result =  add_Function(20,50)
print(result)
```

7-Function with arbitrary positional arguments.

```python
def add_Function(externaMark=20, internalMark=51):
    totalMarkPlus = internalMark + externaMark
    print("My Internal mark is ", internalMark)
    print("My External mark is ", externaMark)
    return totalMarkPlus

result =  add_Function()
print(result)
```

8-Built in Functions

Built-in functions, such as help() to ask for help, min() to get the minimum value, print() to print an object to the terminal

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Function	Description
abs()	Returns the absolute value of a number print(abs(-1)) Output: 1
all()	Returns True if all items in an iterable object are true lst=[10,20.5,"abc",None] print(all(lst)) Output: False
any()	Returns True if any item in an iterable object is true lst=[10,20.5,"abc",None] print(any(lst))

	True
| ascii() | Returns a readable version of an object. Replaces none-ascii characters with escape character<br><br>```python
x = ascii("My name is St å le")
print(x)
```<br>output<br><br>'My name is St \xe5 le'<br><br>---------<br><br>print(ascii('PythØn'))<br>Output:<br>Pyth\xd8n |
| bin() | Returns the binary version of a number<br><br>```python
print(bin(1))
print(bin(72))
```<br>output<br><br>0b1 |

	0b1001000
bool()	Returns the boolean value of the specified object print(bool(0)) print(bool(1)) print(bool(**None**)) print(bool(**False**)) print(bool("**Lin**")) output False True False False True ------------- print(bool(0)) print(bool("bas")) Output:

	False True
bytearray()	Returns an array of bytes print(bytearray(0)) print(bytearray(1)) output bytearray(b'') bytearray(b'\x00') Note:There is no bytesarray () for negative values print(bytearray(-1)) output ValueError: negative count
bytes()	Returns a bytes object print(bytes(0)) print(bytes(1)) print(bytes(5))

	output b'' b'\x00' b'\x00\x00\x00\x00\x00' Note:Bytes is onlyh for encoding characters print(bytes('**A**')) output TypeError: string argument without an encoding
callable()	Returns True if the specified object is callable, otherwise False print("**Is str callable?** ", callable(str)) *# str class* print("**Is len callable?** ", callable(len)) *# len function* print("**Is list callable?** ", callable(list)) *# list class* num=10 print("**Is variable callable?** ", callable(num)) output Is str callable? True Is len callable? True Is list callable? True Is variable callable? False

The `callable()` method works with user-defined classes and functions, as shown below.

```python
class student:
    def greet(self):
        print("Hello there")

std = student()
print("Is student class callable? ",callable(student))
print("Is student.greet() callable? ",callable(std.greet))
print("Is student instance callable? ",callable(std))
```

==output==

Is student class callable? True

Is student.greet() callable? True

Is student instance callable? False

---

chr()

Returns a character from the specified Unicode code.

```python
print(chr(66))
```

==Output:==

B

classmethod()	Converts a method into a class method
compile()	Returns the specified source as an object, ready to be executed
complex()	Returns a complex number cnum=complex(5,6) print(cnum) ==Output:== 5+6j
delattr()	Deletes the specified attribute (property or method) from the specified object
dict()	Returns a dictionary (Array) numdict = dict(I='one', II='two', III='three') print(numdict) ==output== {'I': 'one', 'II': 'two', 'III': 'three'}
dir()	Returns a list of the specified object's properties and methods

```python
print(dir())
print("==========")
print(dir(int))
```

output

['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__']

==========

['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__',

	'__xor__', 'as_integer_ratio', 'bit_count', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
divmod()	Returns the quotient and the remainder when argument1 is divided by argument2 print(divmod(6,2)) print(divmod(8,3)) print(divmod(7,2)) print(divmod(3,19)) output (3, 0) (2, 2) (3, 1) (0, 3)
enumerate()	Takes a collection (e.g. a tuple) and returns it as an enumerate object cities = ['Delhi','Chicago','New York'] for item in enumerate(cities): print(item)

	 (0,"Delhi") (1,"Chicago") (2,"New York")
eval()	Evaluates and executes an expression
exec()	Executes the specified code (or object)
filter()	Use a filter function to exclude items in an iterable object
float()	Returns a floating point number print(float(10)) 10.0
format()	Formats a specified value
frozenset()	Returns a frozenset object
getattr()	Returns the value of the specified attribute (property or method)

globals()	Returns the current global symbol table as a dictionary
hasattr()	Returns True if the specified object has the specified attribute (property/method)
hash()	Returns the hash value of a specified object
help()	Executes the built-in help system
hex()	Converts a number into a hexadecimal value
id()	Returns the id of an object print(id(10)) Output: 157594035292
input()	Allowing user input
int()	Returns an integer number
isinstance()	Returns True if a specified object is an instance of a specified object
issubclass()	Returns True if a specified class is a subclass of a specified object

iter()	Returns an iterator object
len()	Returns the length of an object
list()	Returns a list
locals()	Returns an updated dictionary of the current local symbol table locals() function in Python returns the dictionary of current local symbol table. Symbol table: It is a data structure created by compiler for which is used to store all information needed to execute a program. ... Unlike from globals() this function can not modify the data of local symbol table. https://www.geeksforgeeks.org/python-locals-function/
map()	Returns the specified iterator with the specified function applied to each item
max()	Returns the largest item in an iterable marks = [45,78,34] a = max(marks) print(a)

	<mark>output</mark> 78
[memoryview()](#)	Returns a memory view object
[min()](#)	Returns the smallest item in an iterable
[next()](#)	Returns the next item in an iterable
[object()](#)	Returns a new object
[oct()](#)	Converts a number into an octal
[open()](#)	Opens a file and returns a file object
[ord()](#)	Convert an integer representing the Unicode of the specified character
[pow()](#)	Returns the value of x to the power of y
[print()](#)	Prints to the standard output device
property()	Gets, sets, deletes a property
[range()](#)	Returns a sequence of numbers, starting from 0 and increments by 1 (by default)

repr()	Returns a readable version of an object
reversed()	Returns a reversed iterator
round()	Rounds a numbers
set()	Returns a new set object
setattr()	Sets an attribute (property/method) of an object
slice()	Returns a slice object
sorted()	Returns a sorted list
staticmethod()	Converts a method into a static method
str()	Returns a string object
sum()	Sums the items of an iterator
super()	Returns an object that represents the parent class
tuple()	Returns a tuple
type()	Returns the type of an object

vars()	Returns the __dict__ property of an object
zip()	Returns an iterator, from two or more iterators

9-User defined functions

Python lets us group a sequence of statements into a single entity, called a function. A Python function may or may not have a name.

1. This Python Function help divide a program into modules. This makes the code easier to manage, debug, and scale.
2. It implements code reuse. Every time you need to execute a sequence of statements, all you need to do is to call the function.

3. This Python Function allow us to change functionality easily, and different programmers can work on different functions.

```python
def add_Fucntion(a, b):
    addResult = a + b
    subResult = a * b
    multiResult = a * b

    return addResult,subResult,multiResult

result = add_Fucntion(10,2)
print(result)
```

----

10   Function with arbitrary keyword arguments.
    1. * args
    2. ** kwargs (name and value pair)
11 High Order Functions (MRF)
12 Anonyms Functions / Lambda Functions

13 Recursion functions