Introduction. Function Call

Sunday, January 16, 2022 1:05 AM

1. What is a function?

- A function is a piece of code that you can "call" repeatedly to do one thing.
- Python have both built-in functions and user-defined functions
- We use functions because they save us time and easier for user.

2. Calling Functions

function_name (arguments)

where: argument = value given to a function
pass = to provide an argument to a function
call = ask Python to execute a function (by name)
return = give a value back to where the function was called from

Ex In the example below `abs()` is the **function** we are **calling** and `-20` is the **argument** that we're passing to the function.

In [ ]:
```
x = abs(-20)
print(x)
```

And, `20` is what the function **returns**.

In the example below, you can see we're using two functions calls in the same line of code.

In [ ]:
```
y = abs(-20) + abs(-3)
print(y)
```

Both `print()` and `abs()` are built-in function in Python.

3. Evaluation and Expression

- The assignment statement (=) also applies if RHS is a function
  +) The value of the expression on RHS is figured out
  +) The value is assigned to the variable on LHS

First, function is called while passing it an argument

↓

X = abs (-20)

Then, what the function returns is assigned to x

Built-in Functions. Output. Input

Tuesday, 18 January 2022 21:43

1. Built-in Functions

A list of functions that is already defined by Python

`dir()` → Returns list of the attributes and methods of any object

`pow(base, exp, mod = None)` → equivalent to $base^{exp}$ with 2 arguments
or $base^{exp} \% mod$ with 3 arguments

`int()` → converts a number to an integer - throwing away everything after the decimal
converts a string to an integer if the string contains only number

`help(function_name)` → gives documentation on functions

```
In [9]: help(abs)
```

```
Help on built-in function abs in module builtins:

abs(x, /)
    Return the absolute value of the argument.
```

`abs()` → Returns the absolute value of argument

`ceil()` → Rounds up to nearest integer
`floor()` → Rounds down to nearest integer
`round(variable, number_of_decimals)` → round to specific number of decimals

Built-in Functions

A	E	L	R
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
any()			round()
anext()	F	M	
ascii()	filter()	map()	S
	float()	max()	set()
B	format()	memoryview()	setattr()
bin()	frozenset()	min()	slice()
bool()			sorted()
breakpoint()	G	N	staticmethod()
bytearray()	getattr()	next()	str()
bytes()	globals()		sum()
		O	super()
C	H	object()	
callable()	hasattr()	oct()	T
chr()	hash()	open()	tuple()
classmethod()	help()	ord()	type()
compile()	hex()		
complex()		P	V
	I	pow()	vars()
D	id()	print()	
delattr()	input()	property()	Z
dict()	int()		zip()
dir()	isinstance()		
divmod()	issubclass()		_
	iter()		__import__()

2. Output

`print(argument)` → • Displaying message to user
• The argument can mix different constant and types

print (argument) → • Displaying message to user
• The argument can mix different constant and types
• print() can take more than 1 argument and each comma (,) ⇒ a space

```
In [11]: name = 'Michael'
         age = 20
         print("Hello, my name is", name, "and I am age", age, "years old.")

         Hello, my name is Michael and I am age 20 years old.
```

print (*objects, sep = ' ', end = '\n', file = sys.stdout, flush = False)

sep: define what will the , be

\n → Go down 1 line
\t → Tab

3. Input

input (argument) → For reading inputs from user

The text you want displayed to the user (before space to input)

```
In [*]: name = input("What is your name? ")
        print("Hello, my name is", name)

        1. What is your name?
        2. What is your name? Viet Minh Nguyen
        3. What is your name? Viet Minh Nguyen
           Hello, my name is Viet Minh Nguyen
```

NOTE: The value returned from input is always a string

⇒ For problems requiring users to type in the value, we must include int() to convert or float()

```
In [*]: number = input("Input a number ")
        number + 5

        Input a number 7

        ---------------------------------------------------------------------
        TypeError                 Traceback (most recent call last)
        /tmp/ipykernel_78/1572949926.py in <module>
            1 number = input("Input a number ")
        ----> 2 number + 5

        TypeError: can only concatenate str (not "int") to str
```

✗

```
In [2]: number = input("Input a number ")
        int(number) + 5

        Input a number 7

Out[2]: 12
```

✓

Importing Functions and Modules

Tuesday, 18 January 2022 22:32

- Not all useful functions are built-in → They must be imported
- Groups of functions are stored in separate Python files → modules
  To get access to the functions in a module, you need to import the module

```
# sine or cosine is in module math.
```

Import statement

```
import module_name
```

To access a function within a module:

```
module_name.function_name(arguments)
```

Get a list of the function using the help() function

```
In [4]: print(math.sqrt(16))
```

```
4.0
```

```
In [5]: degrees = 90
        sin90 = math.sin(degrees)
        print(sin90)
```

```
0.8939966636005579
```

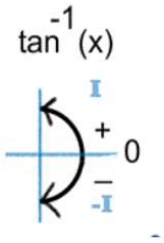
⇒ Trigonometric functions are in radiants

math.radians() → Convert from degrees to radians
math.degrees() → convert from radians to degree

Differences in Arctan Functions

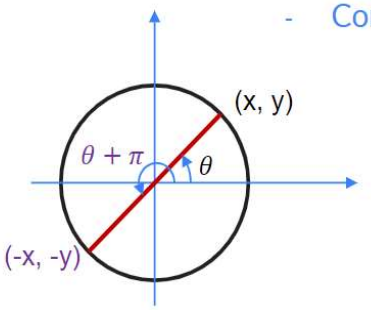
`math.atan(x)`

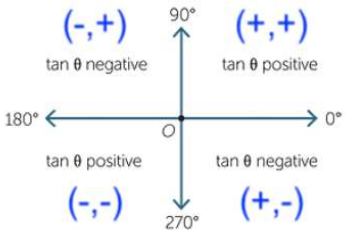
- Single sign is used in the calculation.



`math.atan2(y, x)`

- Both signs used!!
- Computes correct quadrant for the angle.





Example:

Let p be a point with coordinates (x, y) = (-1, -2)

- `math.atan(-2/-1) = 1.107` (two negative signs cancel out !)
- `math.atan2(-2,-1)= -2.03`

Defining Your Own Function

Tuesday, January 18, 2022 11:42 PM

The general form of a function definition is :

```
def function_body (parameters):
    body
    return
```

Indent → ; Colon ← (after the closing parenthesis) ; (parameter 1, 2, ...)

where : def = keyword stands for definition. The def statement must end with a colon.
function_name = name you will use to call the function
parameters = variables that get values when you call the function.
body = sequence of commands like assignments, multiplication, function calls
return = save [statement] (result of function). If it's not, same as return None

⊛ All lines of body must be indented ↳ use in labs. "print" is only for users

In [6]:
```python
def square(x):
    return x*x
```

In [7]:
```python
num = 4
num_sq = square(num)
print(num, num_sq)
```

```
4 16
```

""" DOCSTRING """ (optional)
→ Description of function.
→ Shows what appears when we use help (function_name)

- The parameter (variable) inside the function and the variable outside the function are completely independent ! (just the "holes" to input the values when called)

In [13]:
```python
def square(x):
    print("Inside function. x = ", x)
    return x*x
```

In [14]:
```python
x = 4
print("Outside function. x = ", x)
num_sq = square(8)
print("Outside function. x = ", x)
```

```
Outside function. x =  4
Inside function. x =  8
Outside function. x =  4
```

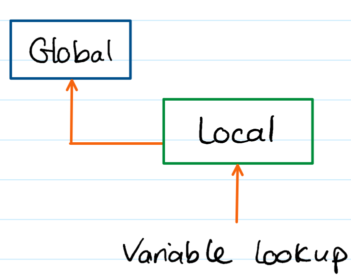
2. Local and Global Scope

A variable is only available from inside the region it is created → variable scope

a) Local Scope

- Whenever you define a variable with a function, its scope lies ONLY within the function
- It is accessible from the point at which it is defined until the end of the function and exists for as long as the function is executing

⇒ Its value cannot be changed or even accessed from outside function



```
def my_function ():
    name = 'Sebastian'
my_function ()
print (name)
```
ERROR

name is local to function and not accessible outside

```
def my_function ():
    print (name)
name = 'Sebastian'
my_function ()
```
→ Sebastian

name is not in function, but is defined in global scope

```
def my_function ():
    name = 'Sebastian'
    print (name)
name = 'Ben'
my_function ()
```
→ Sebastian

name was accesible in the function, and found (Done)

b) Global Scope

- Whenever a variable is defined outside any function, it becomes global variable.
- This means that variables and functions defined outside of a function are also accessible inside of a function.

Design Receipe

Thursday, 20 January 2022 18:52

1. Examples (What do you want your function calls to look like?)
2. Type Contract (Specify the types of parameters and return values)
3. Header (Decide on the name of the function)
4. Description (Write a short description of what the function does)
5. Body (Write the code that actually does the thing that you want)
6. Test (Verify the function using examples)

Nested Functions Calls

Thursday, 20 January 2022 19:09

1. Nested Function Calls

Nested function calls are calling functions within functions ⇒ Normal !

e.g.
```
bigger_area = max (triangle_area (3.8, 7.0), triangle_area (3.5, 6.8))
print (bigger_area)
```
→ 13.3

2. Calling Functions within Functions

```
1  def convert_to_kelvin (degree_f):
2      def convert_to_celcius (degree_f):
3          degree_c = (degree_f - 32)*(5/9)
4          return (degree_c)
5      degree_k = convert_to_celcius (degree_f) + 273
6      print (degree_k)
7
8  convert_to_kelvin (55)
```

Not degree_c, because degree_c is not defined after def of function convert_to_celcius ()