Designing and Using Functions PART-1

Two types of function:

Customized function that programmer design in their code

built-in functions that come with Python, and we'll also show you how to define your own functions.

How to get the list of built-in functions in Python:

```
>>>dir(__builtins__)
```

To get help of any function:

```
>>>help(function_name)
```

Ex:

```
>>>help (max)
```

Practice of some functions and operator:

1. What is the difference between the execution sequence of :

```
>>> 2 + 3 – 2      &
>>>  2**2**3
```

2. What is the difference between

```
>>>2**2**3    &
>>>(2**2)**3
```

3. Is the following function work?

```
>>>4+-2
```

4. How about this function:

```
>>> 2*2+        >>> 1+3)/4        >>> (1+5+
```

5. What is the following function give:

```
>>>1>2    >>>4>=1    >>>2>=2    >>>4+5 <=44    >>> 2+(4<=6)    >>> 2=2
```

Practice of some functions and operator:

6. Write function to check if number is odd/even (for even gives True, odd False)?

7. What is the result of
a) >>>max(4.-3,1)
b) >>>max(3,6,5)
c) >>>max(6, “Sadiq” , 1300)
d) >>>max (“sadiq”, “vida”)
e) >>>second_smallest_vlaue(5,3,22,1)

8. Is this function OK:

```
>>> max( pow(2,2), abs(-5), 3-11)
```

9. Write pow() function to evaluate: 2**2**3

10. You can get the integer part of number via 11//6 ; how to round it to two number of digits
11. What is the difference between >>> X=7 & >>>X==7

Example of variables

1. Use variable
2. Change variable
3. Retrieve variable
4. Mix variables
5. Use unknown variables
6. Know the type of variable

```
>>> x=3
>>> type(x)
<class 'int'>
>>> x='sadiq'
>>> type (x)
<class 'str'>
>>> x=3.4
>>> type(x)
<class 'float'>
```

Algebraic expressions

The Python interactive shell can be used to evaluate algebraic expressions

`14//3` is the quotient when 14 is divided by 3 and `14%3` is the remainder

`2**3` is 2 to the 3rd power

`abs()`, `min()`, and `max()` are functions

- `abs()` takes a number as input and returns its absolute value
- `min()` (resp., `max()`) take an arbitrary number of inputs and return the "smallest" (resp., "largest") among them

```
>>> 2 + 3
5
>>> 7 - 5
2
>>> 2*(3+1)
8
>>> 5/2
2.5
>>> 5//2
2
>>> 14//3
4
>>> 14%3
2
>>> 2**3
8
```

Boolean expressions

In addition to algebraic expressions, Python can evaluate Boolean expressions

- Boolean expressions evaluate to `True` or `False`
- Boolean expressions often involve comparison operators
  <, >, ==, !=, <=, and >=

```
>>> 2 < 3
True
>>> 2 > 3
False
>>> 2 == 3
False
>>> 2 != 3
True
>>> 2 <= 3
True
>>> 2 >= 3
False
>>> 2+4 == 2*(9/3)
True
```

In a an expression containing algebraic and comparison operators:

- Algebraic operators are evaluated first
- Comparison operators are evaluated next

Boolean operators

In addition to algebraic expressions, Python can evaluate Boolean expressions

- Boolean expressions evaluate to `True` or `False`
- Boolean expressions may include Boolean operators `and`, `or`, and `not`

```
>>> 2<3 and 3<4
True
>>> 4==5 and 3<4
False
>>> False and True
False
>>> True and True
True
>>> 4==5 or 3<4
True
>>> False or True
True
>>> False or False
False
>>> not(3<4)
False
>>> not(True)
False
>>> not(False)
True
>>> 4+1==5 or 4-1<4
True
```

In a an expression containing algebraic, comparison, and Boolean operators:

- Algebraic operators are evaluated first
- Comparison operators are evaluated next
- Boolean operators are evaluated last

Representation of numbers in Python

Integers can be represented exactly. There is no limits to their size (other than the size of you memory).

Floating points are approximation of real numbers. Python uses a double-precision standard format (IEEE 754) that can represent real numbers in a range of 10^{-308} to 10^{308} with 16 to 17 digits of precision.

Operations : -, **, *, /, %, //, +, -
<, <=, ==, !=, >, >=

== should be avoided in floats

Number-type operators

Table 2.4 Number-type operators. Listed are the operators that can be used on number objects (e.g., `bool`, `int`, `float`). If one of the operands is a `float`, the result is always a `float` value; otherwise, the result is an `int` value, except for the division (`/`) operator, which always gives a `float` value.

Operation	Description	Type (if `x` and `y` are integers)
`x + y`	Sum	Integer
`x - y`	Difference	Integer
`x * y`	Product	Integer
`x / y`	Division	Float
`x // y`	Integer division	Integer
`x % y`	Remainder of `x // y`	Integer
`-x`	Negative `x`	Integer
`abs(x)`	Absolute value of `x`	Integer
`x**y`	`x` to the power `y`	Integer

Operator Precedence

Operator	Description
`[expressions...]`	List definition
`x[], x[index:index]`	Indexing operator
`**`	Exponentiation
`+x, -x`	Positive, negative signs
`*, /, //, %`	Product, division, integer division, remainder
`+, -`	Addition, subtraction
`in, not in, <, <=, >, >=, <>, !=, ==`	Comparisons, including membership and identity tests
`not x`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR

Naming rules

(Variable) names can contain these characters:

- a through z
- A through Z
- the underscore character _
- digits 0 through 9

Names cannot start with a digit though

For a multiple-word name, use

- either the underscore as the delimiter
- or *camelCase* capitalization

Short and meaningful names are ideal

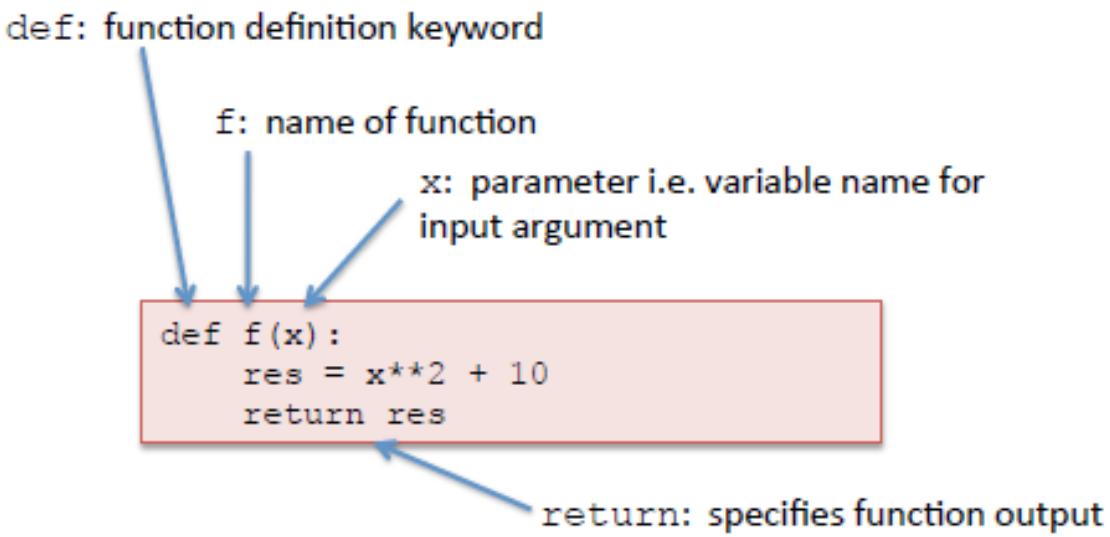
```
>>> My_x2 = 21
>>> My_x2
21
>>> 2x = 22
SyntaxError: invalid syntax
>>> new_temp = 23
>>> newTemp = 23
>>> counter = 0
>>> temp = 1
>>> price = 2
>>> age = 3
```

Defining new functions

A few built-in functions we have seen:

- `abs()`, `max()`, `print()`

New functions can be defined using `def`




```
>>> abs(-9)
9
```

```
>>> abs(-9)
9
>>> max(2, 4)
4
>>> print()

>>> def f(x):
        res = 2*x + 10
        return x**2 + 10


>>> f(1)
11
>>> f(3)
19
>>> f(0)
10
```

Comments and docstrings

Python programs should be documented

- So the developer who writes/maintains the code understands it
- So the user knows what the program does

Comments

```
def f(x):
    res = x**2 + 10    # compute result
    return res         # and return it
```

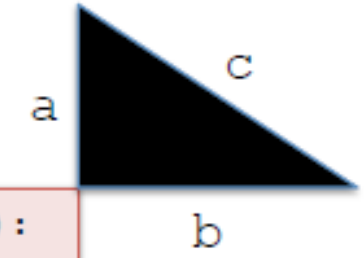
Docstring

```
def f(x):
    'returns x**2 + 10'
    res = x**2 + 10    # compute result
    return res         # and return it
```

```
>>> help(f)
Help on function f in module
__main__:

f(x)

>>> def f(x):
        'returns x**2 + 10'
        res = x**2 + 10
        return res

>>> help(f)
Help on function f in module
__main__:

f(x)
    returns x**2 + 10

>>>
```

Defining new functions

The general format of a function definition is




```
def <function name> (<0 or more variables i.e. parameters>):
    <indented function body>
```

Let's develop function `hyp()` that:

- Takes two numbers as input (side lengths a and b of above right triangle)
- Returns the length of the hypotenuse c

```
>>> hyp(3,4)
5.0
>>>
```

```
import math
def hyp(a, b):
    res = math.sqrt(a**2 + b**2)
    return res
```

print() versus return

```
def f(x):
    res = x**2 + 10
    return res
```

```
>>> f(2)
14
>>> 2*f(2)
28
```

Function returns value of `res` which can then be used in an expression

```
def f(x):
    res = x**2 + 10
    print(res)
```

```
>>> f(2)
14
>>> 2*f(2)
14
Traceback (most recent call last):
  File "<pyshell#56>", line 1, in
<module>
    2*f(2)
TypeError: unsupported operand
type(s) for *: 'int' and
'NoneType'
```

Function prints value of `res` but does not return anything

Designing and Using Functions PART-2

Memory Addresses: How Python Keeps Track of Values

Python used in real time application, uploaded to sensor, EPROM, engineers must trace exactly where each instruction, data is stored for the quality control of the systems.

Python keeps track of each value in a separate object and that each object has a
memory address. You can discover the actual memory address of an object using
built-in function id:

>>> help(id)
Help on built-in function id in module builtins:

How cool is that? Let's try it:

```
>>> id(-9)
4301189552
>>> id(23.1)
4298223160
>>> shoe_size = 8.5
>>> id(shoe_size)
4298223112
>>> fahrenheit = 77.7
>>> id(fahrenheit)
4298223064
```

The addresses you get will probably be different from what's listed above since values get stored wherever there happens to be free space. Function objects also have memory addresses:

```
>>> id(abs)
4297868712
>>> id(round)
4297871160
```

Defining Our Own Functions

The built-in functions are useful but pretty generic.

Often there aren't builtin functions that do what we want, such as calculate mileage or play a specific game.

When we want functions to do these sorts of things, we have to write them ourselves.

Example:

```
>>> def convert_to_celsius(fahrenheit):
... return (fahrenheit - 32) * 5 / 9
```

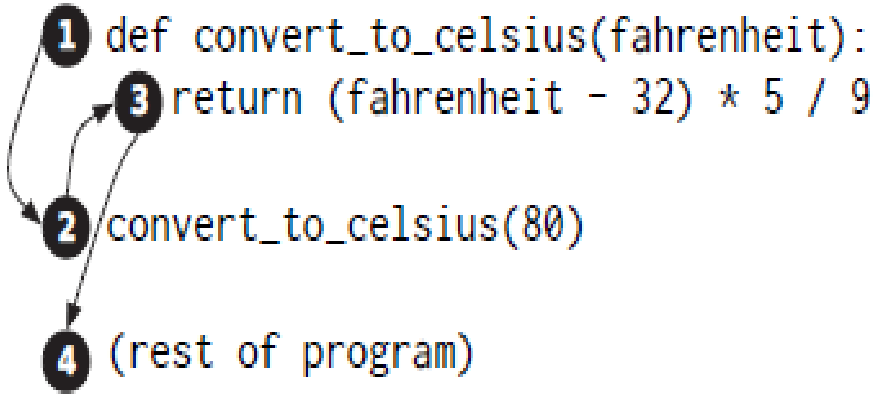
Once we create the function, we can can call help to this function.

```
Help(convert_to_celsious)
```

How Python executes the code

1. Python start executes the function definition
2. Next, Python executes function call convert_to_celsius(212) by assigning 212 f to the parameter
3. Python now executes the return statement. Fahrenheit refers to 212, We use the word return to tell Python what value to produce as the result of the function call, so the result of calling convert_to_celsius(212) is 100.
4. Once Python has finished executing the function call, it returns to the place where the function was originally called.

Here is a picture showing this sequence:



Using Local Variables for Temporary Storage

Complex must be broken down into separate steps.
Easy solution
Clear code
break down the evaluation of the quadratic polynomial *ax2+ bx + c* into several steps
Statement must be be aligned (done by the development environment)

```
>>> def quadratic(a, b, c, x):
... first = a * x ** 2
... second = b * x
... third = c
... return first + second + third
```

first, second, and third that are local variables
created when function is called, and erased when the function returns.

Omitting a Return Statement: None

If you don't have a return statement in a function, nothing is produced:

```
>>> def f(x):
... x = 2 * x
...
>>> res = f(3)
>>> res
>>>
>>> print(res)
None
>>> id(res)
1756120
```

Variable res has a value: it's None! And None has a memory address. If you don't have a return statement in your function, your function will return None.

Exercise

Write function which takes base and height as parameters and return The area of the triangle?

3 mins