

# Functions

Department of Computer Science and Engineering, ITER  
Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar, Odisha, India.



# Contents

- 1 Introduction
- 2 Built-in Functions
- 3 Function Definition and Call
- 4 Importing User-Defined Module
- 5 Assert Statement

# Introduction

- Simple statements can be put together in the form of functions to do useful tasks.
- To solve a problem, divide it into simpler sub-problems.
- The solutions of the sub-programs are then integrated to form the final program.
- This approach to problem solving is called step-wise refinement method or **modular approach**.
- Functions are generally of two types. Such as:
  1. Built-in functions
  2. User-defined functions

# Built-in Functions

- Built-in Functions are predefined functions that are already available in Python.
- The function ***input*** enables us to accept an input string from the user without evaluating its value.
- The function *input* continues to read input text from the user until it encounters a newline. For Example:

```
>>> name = input('Enter a Name:')  
Enter a Name: Alok
```

```
>>> name
```

```
'Alok'
```

Here, the string 'Enter a Name:' specified within the parentheses is called an argument.

- The function *eval* is used to evaluate the value of a string. For Example:

```
>>> eval('15')
```

```
15
```

```
>>> eval('15 + 10')
```

```
25
```

# Built-in Functions (Cont.)

- The value returned by a function may be used as an argument for another function in a nested manner. This is called **composition**.

For Example:

```
>>> n1 = eval(input('Enter a Number:'))
```

Enter a Name: 234

```
>>> n1
```

234

- We can print multiple values in a single call to print function, where expressions are separated by comma. For Example:

```
>>> print(2, 567, 234)
```

2 567 234

```
>>> name = 'Raman'
```

```
>>> print('hello', name, '2+2 =', 2+2)
```

hello Raman 2+2 = 4

# Built-in Functions (Cont.)

- We can use python escape sequences such as `\n` (newline), `\t` (tab), `\a` (bell), `\b` (backspace), `\f` (form feed), and `\r` (carriage return). For Example:

```
>>> print('hello',name, '\n 2+2 =', 2+2)
hello Raman
2+2 = 4
```

- Python function `type` tells us the type of a value. For Example:

```
>>> print(type(12), type(12.5), type('hello'), type(int))
<class'int'>, <class'float'>, <class'str'>, <class'type'>
```

- The `round` function rounds a number up to specific number of decimal places. For Example:

```
>>> print(round(89.625,2), round(89.625), round(89.625,0))
89.62 90 90.0
```

# Built-in Functions (Cont.)

- The input function considers all inputs as strings. Hence, type conversion is required. For Example:

```
>>> str(123)
```

```
'123'
```

```
>>> float(123)
```

```
123.0
```

```
>>> int(123.0)
```

```
123
```

```
>>> str(123.45)
```

```
'123.45'
```

```
>>> float('123.45')
```

```
123.45
```

```
>>> int('123.45') //String incompatible for conversion
```

```
Traceback (most recent call last):
```

```
File "<pyshell#3>", line 1, in <module>
```

```
int('123.45')
```

```
ValueError: invalid literal for int() with base 10: '123.45'
```

# Built-in Functions (Cont.)

- The functions *max* and *min* are used to find maximum and minimum values respectively; can also operate on string values.
- The integer and floating point values are compatible for comparison; whereas numeric values cannot be compared with string values. For Example:

```
>>> max(59,80,95.6,95.2)
```

```
95.6
```

```
>>> min(59,80,95.6,95.2)
```

```
59
```

```
>>> max('hello', 'how', 'are', 'you')
```

```
'you'
```

```
>>> min('hello', 'how', 'are', 'you', 'Sir')
```

```
'Sir'
```

- The function *pow(a,b)* computes *a* to the power *b*.

```
>>> a = pow(2,3)
```

```
>>> a
```

```
8
```



# Built-in Functions (Cont.)

- The function `random` is used to generate a random number in the range  $[0,1)$ . Python module `random` contains this function and needs to be imported for using it.
- Let us agree that player A will play the first turn if the generated falls in the range  $[0,0.5)$ , otherwise player B will play as:

```
if random.random() < 0.5:  
    print('Player A plays the first turn.')
```

else:

```
    print('Player B plays the first turn.')
```

- The `math` module provides some functions for mathematical computations.
- In order to obtain these functions available for use in script, we need to import the `math` module as:  
`import math`
- Name of the module, followed by the separator dot, should precede function name. The `math` module also defines a constant `math.pi` having value 3.141592653589793.

# Built-in Functions (Cont.)

- Functions of math module are:

Function	Description
<code>ceil(x)</code>	Returns the smallest integer greater than or equal to x.
<code>floor(x)</code>	Returns the largest integer less than or equal to x.
<code>fabs(x)</code>	Returns the absolute value of x.
<code>exp(x)</code>	Returns the value of expression $e^{**}x$ .
<code>log(x,b)</code>	Returns the $\log(x)$ to the base b. In the case of absence of the second argument, the logarithmic value of x to the base e is returned.
<code>log10(x)</code>	Returns $\log(x)$ to the base 10. This is equivalent to specifying <code>math.log(x,10)</code> .
<code>pow(x,y)</code>	Returns x raised to the power of y, i.e., $x^{**}y$ .
<code>sqrt(x)</code>	Returns the square root of x.
<code>cos(x)</code>	Returns the cosine of x radians.
<code>sin(x)</code>	Returns the sine of x radians.
<code>tan(x)</code>	Returns the tangent of x radians.

# Built-in Functions (Cont.)

- Functions of math module are (Cont.:

Function	Description
<code>acos(x)</code>	Returns the inverse cosine of x in radians.
<code>asin(x)</code>	Returns the inverse sine of x in radians.
<code>atan(x)</code>	Returns the inverse tangent of x radians.
<code>degrees(x)</code>	Returns a value in degree equivalent of input value x(in radians).
<code>radians(x)</code>	Returns a value in radian equivalent of input value x(in degrees).

- For Example:

```
>>> import math
>>> math.ceil(3.4)
4
>>> math.floor(3.7)
3
```

# Built-in Functions (Cont.)

- If we want to see the complete list the complete list of built-in functions, we can use the built-in function `dir` as `dir(__builtins__)`
- We can get the help on a function as:  

```
>>> import math  
>>> help(math.cos)
```

Help on built-in function cos in module math:

```
cos(...)  
cos(x)
```

Return the cosine of x (measured in radians).

# Function Definition and Call

- The syntax for function definition as:

```
def function_name (comma_separated_list_of_parameters):  
    statements
```

- For Example, We can print a triangle, a blank line and a square as:

```
def main():  
    # To print a triangle  
    print(' *')  
    print(' ***')  
    print(' *****')  
    print('*****')  
    # To print a blank line  
    print()  
    # To print a square  
    print('* * * *')  
    print('* * * *')  
    print('* * * *')  
    print('* * * *')
```

- Python code following colon must be intended, i.e., shifted right.

# Function Definition and Call (Cont.)

- Having developed the function `main` in the script `picture`, we would like to execute it.
- To do this, we need to invoke the function `main` in the following two steps.
  1. In Run menu, click on the option Run Module.
  2. Using Python shell, invoke (call) the function `main` by executing the following command:

```
>>> main()
```

- We can eliminate the need to call function `main` explicitly from the shell, by including in the script `picture`, the following call to function `main`:

```
if '__name__' == '__main__':  
    main()
```

- Python module has a built-in variable called `__name__` containing the name of the module. When the module itself is being run as the script, this variable `__name__` is assigned the string `'__main__'` designating it to be a `__main__` module.

# Function Definition and Call (Cont.)

- Program to print a triangle followed by a square as:

```
def main():  
    # To print a triangle  
    print(' *')  
    print(' ***')  
    print(' *****')  
    print('*****')  
    # To print a blank line  
    print()  
    # To print a square  
    print('* * * *')  
    print('* * * *')  
    print('* * * *')  
    print('* * * *')  
if '__name__' == '__main__':  
    main()
```

Function main serves as a caller function for the callee or called functions triangle and square.

# Function Definition and Call (Cont.)

- A function definition has no effect unless it is invoked.

```
def triangle():  
    # To print a triangle  
    print(' *')  
    print(' ***')  
    print(' *****')  
    print('*****')  
  
def main():  
    # To print a triangle  
    print('Triangle')  
  
if '__name__' == '__main__':  
    main()
```

The script shows that function triangle is not being invoked.



# Function Definition and Call (Cont.)

- **return** statement returns the value of expression following the return keyword. In the absence of the return statement, default value None is returned.
- **Arguments:** Variables or Expressions whose values are passed to called function.
- **Parameters:** Variables or Expressions in function definition which receives value when the function is invoked.
- Arguments must appear in the same order as that of parameters.
- For Example, computing area of a rectangle:

```
def areaRectangle(length, breadth):
```

```
    """
```

```
        Objective: To compute the area of rectangle
```

```
        Input Parameters: length, breadth- numerical value
```

```
        Return Value: area - numerical value
```

```
    """
```

```
        area = length * breadth
```

```
        return area
```

## Fruitful Functions vs void Functions

- A function that returns a value is often called a fruitful function.
- A function that does not return a value is often called a void function.

## Function Help

- Recall that function help can be used to provide a description of built-in functions.
- Function help can also be used to provide description of the function defined by user.
- Function help retrieves first multi-line comment from the function definition

# Function Definition and Call (Cont.)

## Default Parameter Values

- The function parameters may be assigned initial values also called **default values**.
- Function call uses default value, if value for a parameter is not provided.
- Non-default arguments should not follow default arguments in a function definition. For Example:

```
def areaRectangle(length, breadth=1):
```

```
    """
```

```
        Purpose : To compute area of rectangle
```

```
        Input Parameters :
```

```
            length - int
```

```
            breadth (default = 1) - int
```

```
        Return Value: area - int
```

```
    """
```

```
    area = length * breadth
```

```
    return area
```

```
>>> areaRectangle(5)
```

```
5
```

```
>>> areaRectangle(5,2)
```

```
10
```

# Function Definition and Call (Cont.)

## Keyword Arguments

- Python allows us to specify arguments in an arbitrary order in a function call, by including the parameter names along with arguments.
- The syntax for keyword arguments is:  
**parameter\_name = value**
- indeed, in situations involving a large number of parameters, several of which may have default values, keyword arguments can be of great help. For Example:

```
>>> def f(a=2, b=3, c=4, d=5, e=6, f=7, g=8, h=9):  
    return a+b+c+d+e+f+g+h  
>>> f(c=10, g=20)  
62
```

# Importing User-Defined Module

- To access a function from a user-defined module (also known as program or script that may comprise functions, classes, and variables), we need to import it from that module.
- To ensure that the module is accessible to the script, we are currently working on, we append to the system's path, the path to the the folder containing the module.
- The syntax for importing a module:  
**import *name-of-the-module***

# Assert Statement

- The assert statement is used for error checking.
- For Example, When we are going for the calculation of average of marks, be sure that inputs provided by the user are in the correct range.
- For this purpose, we make use of an assert statement that has the following form:  
**`assert condition`**
- If the condition specified in an assert statement fails to hold, Python responds with an assertion error.

# References



Python Programming: A modular approach by Taneja Sheetal, and Kumar Naveen, *Pearson Education India, Inc.*, 2017.