BUILT-IN FUNCTIONS

2.1

Built-in functions are predefined functions that are already available in Python. Recall that we have already used the built-in functions input and print.

input Function

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

invoking input function to take user input

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

>>> name 'Alok‘

The variable name now refers to the string value 'Alok' entered by the user.

So, we say that the function input has been called with the argument 'Enter a number: '.

Further, we say that the function returns the string entered by the user ('Alok') which is subsequently assigned to the variable name.

eval Function

The function eval is used to evaluate the value of a string, for example:

evaluating a string

```
>>> eval('15')          15
>>> eval('15+10')       25
```

composition

The value returned by a function may be used as an argument for another function in a nested manner. This is called composition. To take a numerical value or an expression from a user, we take the input string from the user using the function input, and apply eval function to evaluate its value, for example:

```
>>> n1 = eval(input('Enter a number: '))
                    Enter a number: 234
>>> n1                      234

>>> n2 = eval(input('Enter an arithmetic expression: '))

          Enter an arithmetic expression: 12.0 + 13.0 * 2

>>> n2                              38.0
```

Note that the inputs 234 and 12.0 + 13.0 * 2 were correctly evaluated as int and float values respectively.

print Function

>>> print('hello') hello

Apostrophe marks are used just to tell Python where a string begins and ends and do not form part of the string. Any number of comma-separated expressions may be used while invoking a print function, for example:

printing multiple values in a single call to print function

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

>>> name = 'Raman'

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

Note that when several values are included in a call to the print function separated by commas, they are displayed on the same line, separated by single spaces between them.

It is important to point out that after printing the print control moves to the beginning of the next line.

Thus, the output of a sequence of print function calls appears on separate lines.

Escape Sequence

the output of a single call to the print function is displayed on two lines:

```
>>> print('hello', name, '\n2 + 2 =', 2 + 2)        hello Raman
                                                    2 + 2 = 4
```

\n transfers the print control to the beginning of the next line.

\t which is interpreted as a tab character:

```
>>> print('hello', name, '\t2 + 2 =',2 + 2)
hello Raman  2 + 2 = 4
```

If we do not want Python to interpret an escape sequence, it should be preceded by another backslash. Another way of achieving the same thing is to use R or r before the string containing escape symbol, for example:

```
>>> print('Use \\n for newline')                  Use \n for newline
>>> print(R'Use \n for newline')                 Use \n for newline
>>> print(r'Use \n for newline')                 Use \n for newline
>>> print('Use', R'\n', 'for newline')           Use \n for newline
```

Python also supports various other escape sequences such as

\a (ASCII bell)

\b (ASCII backspace)

\f (ASCII form feed)

\r (ASCII carriage return).

However, some of these are not supported by Python IDLE.

type Function

Values or objects in Python are classified into types or classes, for example, 12 is an integer, 12.5 is a floating point number, and 'hello'is a string. Python function type tells us the type of a value

```
>>> print(type(12), type(12.5), type('hello'), type(int))
```

```
<class 'int'> <class 'float'> <class 'str'>
```

```
<class 'type'>
```

round Function

The round function rounds a number up to specific number of decimal places, for example:

rounding to nearest value

```
>>> print(round(89.625,2), round(89.635), round(89.635,0))
                        89.62  90  90.0

>>> print(round(34.12, 1), round(-34.63))          34.1    -35
```

When calling the function round, the first argument is used to specify the value to be rounded, and the second argument is used to specify the number of decimal digits desired after rounding.

In a call to function round, if the second argument is missing, the system rounds the value of first argument to an integer, for example, round(89.635) yields 90.

Type Conversion

Let the variables costPrice and profit denote cost price and desired profit for a grocery item in a shop. We wish to compute the selling price for the item (Fig. 2.1).

```
1  costPrice = input('Enter cost price: ')
2  profit = input('Enter profit: ')
3  sellingPrice = costPrice + profit
4  print('Selling Price: ', sellingPrice)
```

While executing the above script, if we enter 50 and 5 as the values for costPrice and profit respectively, the system will respond :

undesirable use of + operator

Enter cost price: 50

Enter profit: 5

Selling Price: 505

Note that the input function considers all inputs as strings. Therefore, the value of sellingPrice shown above as 505 is the concatenation of the input values of costPrice and profit, i.e., '50' and '5', respectively. T

o convert the input strings to equivalent integers, we need to use the function int explicitly

conversion from `str` to `int`

```
1  costPrice = int(input('Enter cost price: '))
2  profit = int(input('Enter profit: '))
3  sellingPrice = costPrice + profit
4  print('Selling Price: ', sellingPrice)
```

C

E

Selling Price: 55

We may also use float function if we wish to take decimal value as input from the user. The function str can be used to convert a numeric value to a str value. Next, we give some examples, illustrating the use of some functions for type conversion:

conversion from int to str

```
>>> str(123)                    '123'
>>> float(123)                  123.0
>>> int(123.0)                  123
>>> str(123.45)                 '123.45'
>>> float('123.45')                   123.45
>>> int('123.45')

Traceback (most recent call last):

File "<pyshell#3>", line 1, in <module> int('123.45')

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

conversion from str to float

string incompatible for conversion

Note that last example yields an error since function int cannot be used for converting a string containing decimal value to its integer equivalent.

We already know that the eval function converts the value of the argument to the appropriate type, for example:

>>> eval('50+5') 55

min and max Functions

used to find maximum and minimum value respectively out of several values. These functions can also operate on string values.

```
>>> 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'
```

Note that the integer and floating point values are compatible for comparison. However, numeric values cannot be compared with string values.

pow Function

The function pow(a, b) computes a^b .

Thus, pow(side, 3) gives the side raised to power 3

Random Number Generation

Python provides a function random that generates a random number in the range [0,1).

Python module random contains this function and needs to be imported for using it.

Let us assume that in a game player A will play the first turn if the random number generated falls in the range [0,0.5). Otherwise, player B will play the first turn.

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

The random module provides another function randint that randomly chooses an integer in the specified range;

for example, randint(1,n)

will randomly generate a number in the range 1 to n (both 1 and n included).

Functions from math Module

There are various operations such as floor, ceil, log, square root, cos, and sin that may be required in different applications.

In order to make these functions available for use in a script, we need to import the math module. The import statement serves this purpose:

```
import math
```

Table 2.1 Functions from `math` module

Function	Description
`ceil(x)`	Returns the smallest integer greater than or equal to `x`.
`floor(x)`	Returns the largest integer less than or equal to `x`.
`fabs(x)`	Returns the absolute value of `x`.
`exp(x)`	Returns the value of expression `e**x`.
`log(x,b)`	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.
`log10(x)`	Returns the log(`x`) to the base `10`. This is equivalent to specifying `math.log(x,10)`.
`pow(x,y)`	Returns `x` raised to the power `y`, i.e., `x**y`.
`sqrt(x)`	Returns the square root of `x`.
`cos(x)`	Returns the cosine of `x` radians.
`sin(x)`	Returns the sine of `x` radians.
`tan(x)`	Returns the tangent of `x` radians.

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

The math module also defines a constant math.pi having value 3.141592653589793.

```
>>> import math
>>> math.ceil(3.4)                4
>>> math.floor(3.7)               3
>>> math.fabs(-3)                 3.0
>>> math.exp(2)                   7.38905609893065
>>> math.log(32, 2)               5.0
>>> math.log10(100)               2.0
>>> math.pow(3, 3)                27.0
>>> math.sqrt(65)                 8.06225774829855
>>> math.cos(math.pi)                -1.0
>>> math.sin(math.pi/2)              1.0
>>> math.tan(math.pi/4)              0.999999999999999
>>> math.acos(1)                  0.0
>>> math.asin(1)                  1.570796326794896
>>> math.atan(1)                  0.7853981633974483
>>> math.degrees(math.pi)            180.0
>>> math.radians(180)                3.141592653589793
```

Complete List of Built-in Functions

If we want to see the complete list of built-in functions, we can use the built-in function dir as dir(builtins). To know the purpose of a function and how it is used, we may make use of the function help, for example, to display help on cos function in the math module, we may use

```
>>> 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).
```

```
*
***
*****
*******

*   *   *   *
*   *   *   *
*   *   *   *
*   *   *   *
```

```
01  def main():
02      # To print a triangle
03      print('   *')
04      print('  ***')
05      print(' *****')
06      print('*******')
07
08      # To print a blank line
09
10      print()
11
12      # To print a square
13      print('* * * *')
14      print('* * * *')
15      print('* * * *')
16      print('* * * *')
```

In this program, line number 2 begins with # (hash). This line is a comment

comments enhance readability of the code

side of an equilateral triangle

single line comments start with # is a comment in the statement,

side = 6 # side of an equilateral triangle

docstring (documentation string) may be used for several lines of comment

line 1 function definition --- keyword def, followed by the name of the function main, empty parenthesis, and a colon.

statements (lines 2–16) that form the body of the function main do not begin in column number 1, but begin with four spaces. This is called indentation.

We have used four spaces for indentation as per advice in Google's coding guidelines. However, one may choose a different number of spaces

Python insists on strict indentation rules

def function_name
(comma_separated_list_of_parameters):

a function_name should not be a Python keyword.

The statements inside the function have to be indented from the left margin

It is important to emphasize that apart from the fact that indented code looks elegant, it is also a requirement of Python that the code following a colon must be indented.

Having developed the function main in the script picture, we can execute it by invoking the function main.

>>> 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()

Invoking the function main in the script.

The revised script is shown in Fig. 2.5. Every 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. The `__main__` module, i.e. script `picture` in this case (Fig. 2.5), comprises:

```
01  def main():
02      # To print a triangle
03      print('   *')
04      print('  ***')
05      print(' *****')
06      print('*******')
07
08      # To print a blank line
09
10      print()
11
12      # To print a square
13      print('* * * *')
14      print('* * * *')
15      print('* * * *')
16      print('* * * *')
17
18  if __name__=='__main__':
19      main()
```

When a script is executed, Python creates a run-time environment, called global frame. In the script `picture`, when the definition of the function `main` is encountered, Python makes a note of its definition in the global frame. Next, on encountering the `if` statement, Python checks whether the name of the current module is `__main__`. This being true, the expression `__name__ == '__main__'` evaluates as `True`, and the function `main` is invoked. The check `__name__ == '__main__'` is performed to prevent the accidental calling of a function from an imported module. The `if` statement is used in a script to specify the starting point of execution for the module being run. So, if we have a code that should only be executed when the module is run, and not when it is imported, we need to execute the code using `if` clause as shown in lines 18-19. For the module being imported, variable `__name__` contains the name of imported module.

We can make the above program more elegant by first developing independent functions to print a square and a triangle and then making use of these functions in the main function to print a picture that comprises a triangle and a square separated by a blank line.

```
01  def triangle():
02      # To print a triangle
03      print('   *')
04      print('  ***')
05      print(' *****')
06      print('*******')
07
08  def square():
09      # To print a square
10      print('* * * *')
11      print('* * * *')
12      print('* * * *')
13      print('* * * *')
14
15  def main():
16      # To print a triangle
17      triangle()
18      print()
19      # To print a square
20      square()
21
22  if __name__=='__main__':
23      main()
24  print('\nEnd of program')
```

definitionof functions triangle, square and main

the execution of the if statement (line 22), the control is transferred to the function main (line 15).

Line 16 being a comment, is ignored

In line 17, the function triangle is called

As the main function calls the function triangle, it is said to be the caller function, and the function triangle that is called is said to be the callee function or called function.

function main serves as a caller function for the called functions triangle and square

On completing execution of the function triangle, the control is transferred to the statement immediately following the one that called the function triangle, i.e. at line 18 that prints a blank line.

In line 20, we call the function square. On execution of the body of function square, the control returns to main function (line 21). call to print function at line 24 in the global frame, which now executes, and the program comes to an end.

caller and the called functions need to share some information

To demonstrate this, we develop a program to print the area of a rectangle that takes length and breadth of the rectangle as inputs from the user.

```
1  def areaRectangle(length, breadth):
2      '''
3      Objective: To compute the area of rectangle
4      Input Parameters: length, breadth - numeric value
5      Return Value: area - numeric value
6      '''
7      area = length * breadth
8      return area
```

```
def areaRectangle(length, breadth):
    '''
    Objective: To compute the area of rectangle
    Input Parameters: length, breadth - numeric value
    Return Value: area - numeric value
    '''
    area = length * breadth
    return area

def main():
    '''
    Objective: To compute the area of rectangle based on user input
    Input Parameter: None
    Return Value: None
    '''
    print('Enter the following values for rectangle:')
    lengthRect = int(input('Length : integer value: '))
    breadthRect = int(input('Breadth : integer value: '))
    areaRect = areaRectangle(lengthRect, breadthRect)
    print('Area of rectangle is', areaRect)

if __name__=='__main__':
    main()
print('\nEnd of program')
```

The variables and expressions whose values are passed to the called function are called arguments. At the point of cal to function areaRectangle in line 19, values of arguments lengthRect and breadthRect are passed to parameters length and breadth, respectively. These values are used inside the function areaRectangle, for computing area. While the parameters length and breadth are called formal parameters, or dummy arguments, lengthRect and breadthRect used for invoking the function areaRectangle are called actual parameters, or arguments. The arguments in call to the function must appear in the same order as tha of parameters in the function definition.

arguments: variables/expressions whose values are passed to called function parameters: variables/expressions in function definition which receives value when the function is invoked

the correspondence between arguments and parameters is sometimes called a memory map.

```
01  def areaRectangle(length, breadth):
02      '''
03      Objective: To compute area of rectangle
04      Input Parameters: length, breadth - numeric value
05      Return Value: area - numeric value
06      '''
07      area = length * breadth
08      return area
09
10  def areaSquare(side):
11      '''
12      Objective: To compute area of square
13      Input Parameter: side - numeric value
14      Return Value: area - numeric value
15      '''
16      area = areaRectangle(side, side)
17      return area
18
19  def main():
20      '''
21      Objective: To compute area of rectangle and square based on
22      user input
23      Input Parameter: None
24      Return Value: None
25      '''
26      print('Enter the following values for rectangle:')
27      lengthRect = int(input('Length : integer value: '))
28      breadthRect = int(input('Breadth : integer value: '))
29      areaRect = areaRectangle(lengthRect, breadthRect)
30      print('Area of rectangle is', areaRect)
31      sideSqr = int(input('Enter side of square: integer
        value: '))
32      areaSqr = areaSquare(sideSqr)
33      print('Area of square is', areaSqr)
34
35  if __name__=='__main__':
36      main()
```

Fruitful Functions vs Void Functions

A function that returns a value is often called a fruitful function, for example, the built-in function sin, and abs, and the functions areaRectangle, and areaSquare defined earlier

A function that does not return a value is called a void function, for example, the built-in function print, and the functions triangle, and square, defined earlier

Function help

function help can be used to provide a description of built in functions. It can also be used to provide description of the user defined function if the function has a multi-line comment in it.

function help retrieves first multi-line comment from the function definition

If there are more than one multi-line comments, only the first multi-line comment is displayed.

```
01  def areaRectangle(length, breadth):
02      '''
03      Objective: To compute area of rectangle
04      Input Parameters: length, breadth - numeric value
05      Return Value: area - numeric value
06      '''
07      area = length * breadth
08      return area
09
10  def areaSquare(side):
11      '''
12      Objective: To compute area of square
13      Input Parameter: side - numeric value
14      Return Value: area - numeric value
15      '''
16      area = areaRectangle(side, side)
17      return area
18
19  def main():
20      '''
21      Objective: To compute area of rectangle and square based on
22      user input
23      Input Parameter: None
24      Return Value: None
25      '''
26      print('Enter the following values for rectangle:')
27      lengthRect = int(input('Length : integer value: '))
28      breadthRect = int(input('Breadth : integer value: '))
29      areaRect = areaRectangle(lengthRect, breadthRect)
30      print('Area of rectangle is', areaRect)
31      sideSqr = int(input('Enter side of square: integer
        value: '))
32      areaSqr = areaSquare(sideSqr)
33      print('Area of square is', areaSqr)
34
35  if __name__=='__main__':
36      main()
```

- On executing the command help(areaRectangle), contents specified using multi-line comment will be retrieved.
- >>> help(areaRectangle)
- Help on function areaRectangle in module main : areaRectangle(length, breadth)
- Objective: To compute area of rectangle
- Input Parameters: length, breadth - numeric value Return Value: area - numeric value

Default Parameter Values

The function parameters may be assigned initial values also called default values
When the function areaRectangle is called without specifying the second argument breadth, default value 1 is assumed for it

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

```
01  def areaRectangle(length, breadth = 1):
02      '''
03      Purpose: To compute area of rectangle
04      Input Parameters:
05          length  - int
06          breadth (default 1) - int
07      Return Value: area - int
08      '''
09      area = length * breadth
10      return area
```

However, if the default parameters are specified in a function call, the default values are ignored, for example:

```
>>> areaRectangle(5,2)
10
```

It is important to mention that the default parameters must not be followed by non-default parameters, for example:

```
>>> def areaRectangle(length = 10, breadth):
    return length * breadth
SyntaxError: non-default argument follows default
argument
```

Keyword Arguments

the order of arguments always matched the parameters in the function definition. However, Python allows us to specify arguments in an arbitrary order in a function call, by including the parameter names along with arguments. The arguments specified as

parameter_name = value

syntax for keyword arguments are known as keyword arguments. For example call to the function areaRectangle, the order of arguments is different from the one in the function definition.

areaRect = areaRectangle(breadth = 2, length = 5)

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

access a function from a user-defined module

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 folder containing the module. Once this done, we can import the module by using an instruction like:

specifying system path

import name-of-the-module

Once this is done, we can access all the functions defined in it by using the following notation: module name, followed by a dot, followed by the function name

```
01    import sys
02    sys.path.append('F:\PythonCode\Ch02')
03    import area
04
05    def main():
06        '''
07        Purpose: To compute area of floor
08        Input Parameter: None
09        Return Value: None
10        '''
11        print('Enter the following values for floor')
12        length = int(input('Length: '))
13        breadth = int(input('Width: '))
14        print(area.areaRectangle(length, breadth))
15
16    if __name__=='__main__':
17        main()
```

ASSERT STATEMENT--need to make sure that inputs provided by the user are in the correct range.

If the assertions in lines 18 and 20 hold, the function displays percentage as the output. However, if these assertions fail to hold the system responds with an assertion error, for example:

Enter maximum marks: 150 Enter marks obtained: 155

Traceback (most recent call last):

File "F:/PythonCode/Ch02/percent.py", line 25, in <module>

main()

File "F:/PythonCode/Ch02/percent.py", line 20, in main

assert marks >=0 and marks <=maxMarks AssertionError

```
01  def percent(marks, maxMarks):
02      '''
03      Objective: To find percentage of marks obtained in a subject
04      Input Parameters: marks, maxMarks - float
05      Return Value: percentage - float
06      '''
07      percentage =  (marks / maxMarks) * 100
08      return percentage
09
10  def main():
11      '''
12      Objective: To find percentage of marks obtained in a subject
13      based on user input
14      Input Parameter: None
15      Return Value: None
16      '''
17      maxMarks = float(input('Enter maximum marks: '))
18      assert maxMarks >=0 and maxMarks <=500
19      marks = float(input('Enter marks obtained: '))
20      assert marks >=0 and marks <=maxMarks
21      percentage = percent(marks, maxMarks)
22      print('Percentage is : ', percentage)
23
24  if __name__=='__main__':
25      main()
```

COMMAND LINE ARGUMENTS

Whenever, we execute a script from command line, it takes name of the script as the first argument followed by other input arguments (if any) in string form and stores them in the list sys.argv.

We access the arguments stored in argv using indexes argv[0], argv[1], argv[2], etc.

```
01  import sys
02  def areaRectangle(length, breadth):
03      '''
04      Objective: To compute the area of rectangle
05      Input Parameters: length, breadth - numeric value
06      Return Value: area - numeric value
07      '''
08      area = length * breadth
09      return area
10
11  def main():
12      '''
13      Objective: To compute the area of rectangle based on user input
14      taken as command line arguments
15      Input Parameter: None
16      Return Value: None
17      '''
18      if len(sys.argv) == 3:
19         lengthRect = int(sys.argv[1])
20         breadthRect = int(sys.argv[2])
21         areaRect = areaRectangle(lengthRect, breadthRect)
22         print('Area of rectangle is', areaRect)
23      else:
24         print('Unexpected number of command line arguments!')
25
26  if __name__=='__main__':
27      main()
28  print('\nEnd of program')
```

Since, the number of arguments including the script name should be three, we ensure this using condition len(sys.argv) == 3.

We also assume that in the command usually whitespace(s) are used for separating the command line arguments from each other

python area1.py 20 10

command line arguments length and breadth that follow the script name (area1.py) are stored in argv[1] and argv[2] respectively.