Chapter - 6 : Functions

Approach of Problem Solving

There are three general approaches to writing a program:

1. **Top down** - In the top down approach one starts with the toplevel routine and move down to the low level routine.
2. **Bottom up** - The bottomup approach works in the opposite direction on begins with the specific routines, build them into progressively more complex structures, and end at the top level routine.
3. **Ad hoc** - The ad hoc approach specifies no predetermined method.

**Top-down approach**

- A top down approach also helps one to clarify the overall structure and operation of the program before one code the low level functions.
- The top down method starts with a general description and works towards specifics.
- A good way to design a program is to define exactly what the program is going to do at the top level.
- Each entry in the list should contain only one functional unit.
- A functional unit can be thought of as a black box that performs a single task. Modular programming
- Modular programming is a style that adds structure and readability to the program code.
- It may not make much difference on small projects, but as one starts to work on something bigger it can make the code much easier to read and maintain.
- Structuring the code is a simple task of splitting the program into manageable part so that each part is self contained.
- By creating these self contained modules, one can focus on programming each part.

**Functions**

- A function is a named, independent section of Python code that performs a specific task and optionally returns a value to the calling program.
- A function is named. Each function has a unique name.
- By using the name in another part of the program, one can execute the statements contained in the function. **This is known as calling the function.**
- A function can be called from within any other function.
- A function is independent.
- A function can perform its task without interference from or interfering with other parts of the program.
Defining a Function

- Function blocks begin with the keyword `def` followed by the function name and parentheses `()`.  
- Any input parameters or arguments should be placed within these parentheses. We can also define parameters inside these parentheses.  
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.  
- The code block within every function starts with a colon (`:`) and is indented.  
- The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax

```
def function_name( parameters ) :  
    "function_docstring"
    function_local_variable
    function_statements
    return [expression]
```

Example

```python
# Function definition is here
def printme( str ):
    "This prints a passed string into this function"   # docstring
    print (str)
    return

# Now you can call printme function
printme("This is first call to the user defined function!")
printme("Again second call to the same function")
```

Output

This is first call to the user defined function!
Again second call to the same function
### Built-in Python Functions

<table>
<thead>
<tr>
<th>SNo</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>input()</td>
<td>Allowing user input</td>
</tr>
<tr>
<td>2.</td>
<td>print()</td>
<td>Prints to the standard output device</td>
</tr>
<tr>
<td>3.</td>
<td>int()</td>
<td>Returns an integer number</td>
</tr>
<tr>
<td>4.</td>
<td>float()</td>
<td>Returns a floating point number</td>
</tr>
<tr>
<td>5.</td>
<td>list()</td>
<td>Returns a list</td>
</tr>
<tr>
<td>6.</td>
<td>dict()</td>
<td>Returns a dictionary (Array)</td>
</tr>
<tr>
<td>7.</td>
<td>set()</td>
<td>Returns a new set object</td>
</tr>
<tr>
<td>8.</td>
<td>str()</td>
<td>Returns a string object</td>
</tr>
<tr>
<td>9.</td>
<td>tuple()</td>
<td>Returns a tuple</td>
</tr>
<tr>
<td>10.</td>
<td>type()</td>
<td>Returns the type of an object</td>
</tr>
<tr>
<td>11.</td>
<td>len()</td>
<td>Returns the length of an object</td>
</tr>
<tr>
<td>12.</td>
<td>format()</td>
<td>Formats a specified value</td>
</tr>
<tr>
<td>13.</td>
<td>abs()</td>
<td>Returns the absolute value of a number</td>
</tr>
<tr>
<td>14.</td>
<td>eval()</td>
<td>Evaluates and executes an expression</td>
</tr>
<tr>
<td>15.</td>
<td>round()</td>
<td>Rounds a numbers</td>
</tr>
<tr>
<td>16.</td>
<td>max()</td>
<td>Returns the largest item in an iterable</td>
</tr>
<tr>
<td>17.</td>
<td>min()</td>
<td>Returns the smallest item in an iterable</td>
</tr>
<tr>
<td>18.</td>
<td>oct()</td>
<td>Converts a number into an octal</td>
</tr>
<tr>
<td>19.</td>
<td>pow()</td>
<td>Returns the value of x to the power of y</td>
</tr>
<tr>
<td>20.</td>
<td>range()</td>
<td>Returns a sequence of numbers, starting from 0 and increments by 1 (by default)</td>
</tr>
</tbody>
</table>

#### 1. input() Function

The `input()` function allows user input.

```
input(prompt)
```

```python
x = input('Enter your name: ')
```
2. print() Function

- The print() function prints the specified message to the screen, or other standard output device.
- The message can be a string, or any other object, the object will be converted into a string before written to the screen.

`print(object(s), sep=separator, end=end, file=file, flush=flush)`

<table>
<thead>
<tr>
<th>object(s)</th>
<th>Any object, and as many as you like. Will be converted to string before printed</th>
</tr>
</thead>
<tbody>
<tr>
<td>sep='separator'</td>
<td>Optional. Specify how to separate the objects, if there is more than one. Default is ''</td>
</tr>
<tr>
<td>end='end'</td>
<td>Optional. Specify what to print at the end. Default is '\n' (line feed)</td>
</tr>
<tr>
<td>file</td>
<td>Optional. An object with a write method. Default is sys.stdout</td>
</tr>
<tr>
<td>flush</td>
<td>Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False</td>
</tr>
</tbody>
</table>

```python
print("Hello", "how are you?")
x = ("apple", "banana", "cherry")
print(x)
print("Hello", "how are you?, sep="---")
```

3. int() Function

The int() function converts the specified value into an integer number.

Syntax : `int(value, base)`

- value A number or a string that can be converted into an integer number
- base A number representing the number format. Default value: 10

```python
x = int("12")
print(x+10)
```

3. float() Function

The float() function converts the specified value into a floating point number..

Syntax : `float(value)`

```python
x = float(3)
print(x)
x = float("3.500")
print(x)
```
4. **list() Function**:
   The `list()` function creates a list object.
   A list object is a collection which is ordered and changeable.

5. **dict() Function**:
   The `dict()` function creates a dictionary.
   A dictionary is a collection which is unordered, changeable and indexed.

6. **set() Function**
   The `set()` function creates a set object.
   The items in a set list are unordered, so it will appear in random order.

7. **str() Function**
   The `str()` function converts the specified value into a string.

8. **tuple() Function**
   The `tuple()` function creates a tuple object.
   We cannot change or remove items in a tuple.

*Note – Function list(), dict(), set(), str() & tuple() already covered in Chapter -5.*

9. **type() Function**
   The `type()` function returns the type of the specified object.

   ```python
   a = ('apple', 'banana', 'cherry')
   b = "Hello World"
   c = 33
   
   x = type(a)
   y = type(b)
   z = type(c)
   ```

10. **len() Function**
    The `len()` function returns the number of items in an object.
    When the object is a string, it returns the number of characters in the string.

    ```python
    mylist = ["apple", "banana", "cherry"]
    x = len(mylist)
    print(x)
    
    mylist = "Hello"
    x = len(mylist)
    print(x)
    ```
11. \textit{format()} Function

The \textit{format()} function formats a specified value into a specified format.

\texttt{format(value, format)}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{value}</td>
<td>A value of any format</td>
</tr>
<tr>
<td>\texttt{format}</td>
<td>The format you want to format the value into. Legal values:</td>
</tr>
<tr>
<td></td>
<td>\texttt{&lt;} - Left aligns the result (within the available space)</td>
</tr>
<tr>
<td></td>
<td>\texttt{&gt;} - Right aligns the result (within the available space)</td>
</tr>
<tr>
<td></td>
<td>\texttt{^} - Center aligns the result (within the available space)</td>
</tr>
<tr>
<td></td>
<td>\texttt{=} - Places the sign to the left most position</td>
</tr>
<tr>
<td></td>
<td>\texttt{+} - Use a plus sign to indicate if the result is positive or negative</td>
</tr>
<tr>
<td></td>
<td>\texttt{-} - Use a minus sign for negative values only</td>
</tr>
<tr>
<td></td>
<td>\texttt{'} - Use a leading space for positive numbers</td>
</tr>
<tr>
<td></td>
<td>\texttt{,} - Use a comma as a thousand separator</td>
</tr>
<tr>
<td></td>
<td>\texttt{'} - Use an underscore as a thousand separator</td>
</tr>
<tr>
<td></td>
<td>\texttt{b} - Binary format</td>
</tr>
<tr>
<td></td>
<td>\texttt{c} - Converts the value into the corresponding unicode character</td>
</tr>
<tr>
<td></td>
<td>\texttt{d} - Decimal format</td>
</tr>
<tr>
<td></td>
<td>\texttt{e} - Scientific format, with a lower case e</td>
</tr>
<tr>
<td></td>
<td>\texttt{E} - Scientific format, with an upper case E</td>
</tr>
<tr>
<td></td>
<td>\texttt{f} - Fix point number format</td>
</tr>
<tr>
<td></td>
<td>\texttt{F} - Fix point number format, upper case</td>
</tr>
<tr>
<td></td>
<td>\texttt{g} - General format</td>
</tr>
<tr>
<td></td>
<td>\texttt{G} - General format (using a upper case E for scientific notations)</td>
</tr>
<tr>
<td></td>
<td>\texttt{o} - Octal format</td>
</tr>
<tr>
<td></td>
<td>\texttt{x} - Hex format, lower case</td>
</tr>
<tr>
<td></td>
<td>\texttt{X} - Hex format, upper case</td>
</tr>
<tr>
<td></td>
<td>\texttt{n} - Number format</td>
</tr>
<tr>
<td></td>
<td>\texttt{%} - Percentage format</td>
</tr>
</tbody>
</table>

\texttt{x = format(0.5, \texttt{\%})} \quad \text{Output - 50.000000\%}

\texttt{x = format(255, \texttt{x})} \quad \text{Output - ff}