

# Introduction to Programming

## Python Lab 9: Functions

# Getting Started

- Create a new folder in your disk space with the name **PythonLab9**
- Launch the Python Integrated Development Environment (IDLE) - begin with the **Start** icon in the lower left corner of the screen.
- If you are in a DCSIS laboratory, search using the keyword **Python** and click on **IDLE (Python 3.6 64-bit)**

A window with the title **Python 3.6.2 Shell** should appear. This window is the **Shell**.

## Getting Started (2)

- If you are in the ITS laboratory MAL 109, then right mouse click on the **Start** icon in the lower left corner of the screen.

A list of menu options should appear and click on **Search**. Type **Python** in the search text box at the bottom of the pop-up window. A list of Apps should appear and select

### ***Python 3.4 IDLE(PythonGUI)***

A window with the title **Python 3.4.3 Shell** should appear. This window is the **Shell**.

- In the **Shell** click on **File**. A drop down menu will appear. Click on **New File**. A window with the `title` **Untitled** should appear. This window is the **Editor**.

## Getting Started (3)

- In the Editor, click on **File**, and then in the drop down menu click on **Save As...** .

A window showing a list of folders should appear.

- To search any folder on the list, double click on the folder.
- Find the folder `PythonLab9` and double click on it.
- In the box **File name** at the bottom of the window
  1. Type `CompoundInterest.py`
  2. Then click on the button **Save** in the lower right corner of the window.

The title of the Editor should change to show the location of the file `CompoundInterest.py`.

# Objectives of the exercises set

- Create your own user-defined **functions** in your program.  
When defining a function, you provide a **name** for the function and a **parameter variable** for each argument.

For example, the definition of the function `cubeVolume` is shown below.

```
def cubeVolume ( sideLength ) :  
    volume = sideLength ** 3  
    return volume
```

Function header

Name of function

Name of parameter variable

Put a colon here.

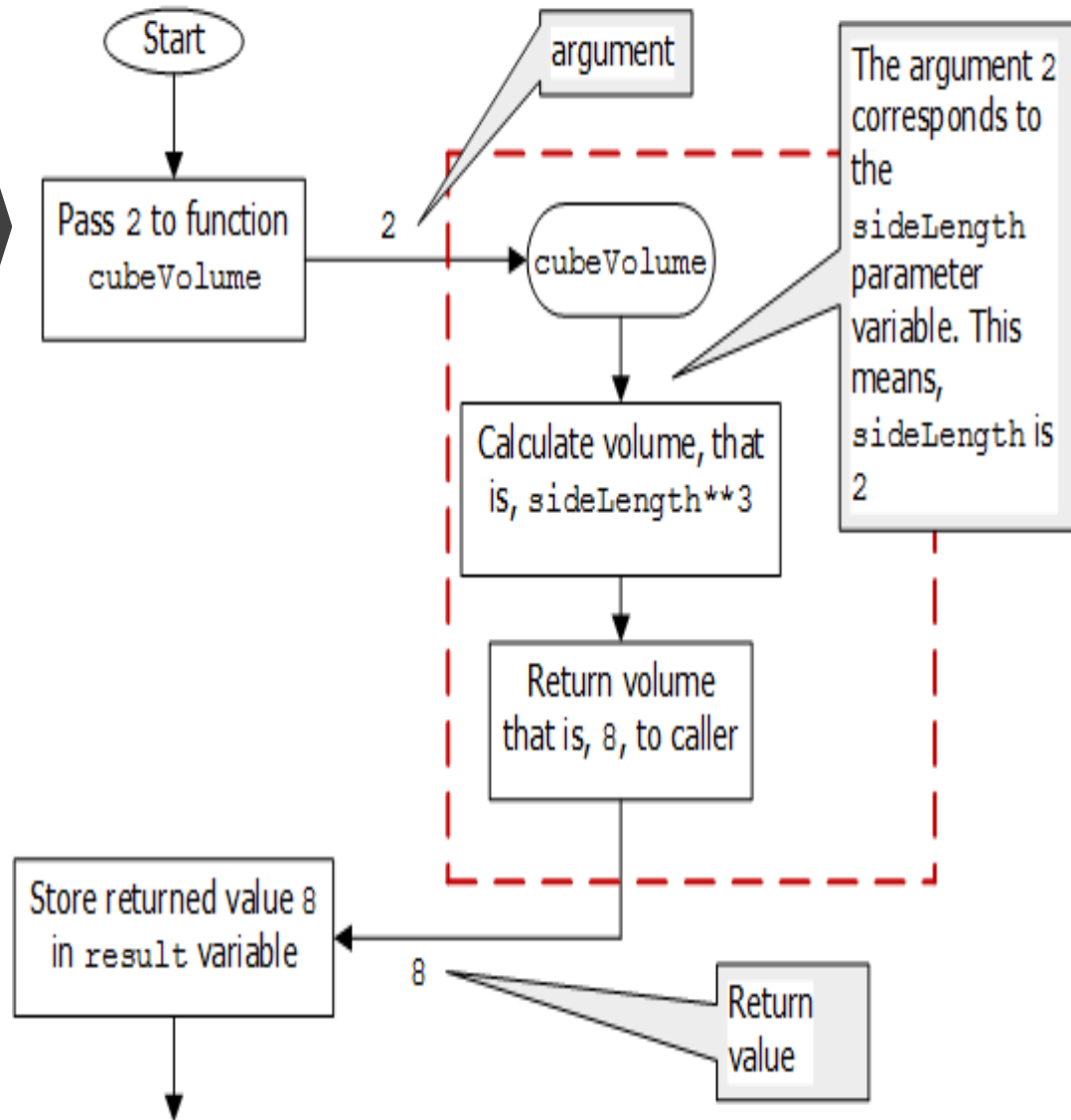
Function body, executed when the function is called. The statements in the body should be indented to the same level.

return statement exits the function and returns the result (that is, the value of `volume`).

```
result = cubeVolume (2)
```

After the definition of `cubeVolume`, call the function with an argument of 2 to calculate the volume, and then store the returned value 8 in the variable `result`.

# Execution flow of a function call to cubeVolume



# Objectives of the exercises set (2)

- Use a `for` statement to implement count-controlled loops that iterate over a range of integer values.

For example,

```

for i in range( 1, 5, 2 ) :
    print(i) #loop body
  
```

The **start** value

The **ending** value is never included in the sequence.

The third argument is the **step** value.

Working of the loop:

value of <code>i</code>	Output using <code>print(i)</code>
1	1
3	3

## Recall

- The `range` function generates a sequence of integers over which the loop iterates.
- The variable `i` is set, at the beginning of each iteration, to the next integer in the sequence generated by the `range` function.

# Objectives of the exercises set (3)

- Use the string format operator **%** and sensible format specifiers to specify how values should be formatted in the output.

For example,

```
print("Quantity: %d Cost: %6.2f" % (quantity, cost))
```

Format specifiers

**Recall:** The format string can contain one or more format specifiers and literal characters. Use the letter `d` for an integer value, and the letter `f` for a floating-point value.

**Recall:** The values/variables to be formatted are defined inside the brackets. Each value replaces one of the format specifiers in the resulting string.



# Program CompoundInterest.py:

## function `balance`

- **Question 2: Problem statement**

Write a function with the name `balance` and with the following three arguments: `initialBalance`, `rate` and `numberYears`. The function `balance` returns the balance in an account given the initial balance, the number of years that have elapsed and the interest rate.

It is assumed that the interest is compounded. Recall that the balance in the account is given by the formula



```
initialBalance * ((1+rate/100)**numberYears)
```

Provide a comment to explain the purpose of the function `balance`. Also include in your comment the types of the parameter values, the type of the returned value, your name and the date.

Save the function `balance` in the file `CompoundInterest.py`.

# Program CompoundInterest.py: function `balance` (2)

- **Problem solving** - Convert the pseudocode below into a sequence of Python statements in your program.

1. Define a function named `balance` as shown below:

```
def balance(initialBalance, rate, numberYears):
```

2. Create an assignment statement to calculate the final balance using the formula below, and store the result in a variable named `finalBalance`. Indent the statement.

```
    initialBalance*((1+rate/100)**numberYears)
```

3. Add a `return` statement to return the result of the function, namely `finalBalance`, in the function body. The `return` and the assignment statement in step 2 have the same indentation.

4. Add a `print` statement to **call** the function `balance`, for example

```
print(balance(100, 6, 2))
```

Align the function header `def` and the `print` statement to the same level of indentation.

Function body, executed when function is called.

The print statement is outside the function definition.

# Program CompoundInterest.py: function `balance` (3)

- Save the function `balance` in the file `CompoundInterest.py`, and then run the program to test it.

# Program CompoundInterest.py: function `balance` (4)

👉 **Note:** You should define the function `balance` **before** you call it later in the program.

## Inputs to the function `balance`

After creating the definition of the function `balance`, call `balance` with the following arguments:

```
balance(100, 6, 2)
```

## Computation in the function `balance`

### Initializing function parameter variables

`initialBalance` is `100`

`rate` is `6`

`numberYears` is `2`

### Function body:

```
finalBalance =  
initialBalance * ((1 + rate / 100)  
** numberYears)  
return finalBalance
```

## Result returned by the function

Modify the `print` statement (in step 4, page 10) to format the result to 2 decimal places along with a suitable format string as shown in the output below.

```
Final balance:  
112.36
```

# Program CompoundInterest.py: function `table`

- **Question 3: Problem statement**

Write a function `table` that requests an initial balance and a number of years and then prints out the two requested inputs together with some text to specify the meaning of the printout. For example, the initial balance could be printed out in the form



```
The initial balance is 100 ukp
```

The function `table` then prints out the final balance for the values -6%, -3%, 0%, 3%, 6% of the rate, given the requested initial balance and the requested number of years. Each value of the rate corresponds to a single line in the print out.

For example, if the rate is -6% and the calculated balance is 126.7894 ukp, then the corresponding line of the printout is



```
rate: -6%, balance: 126.79 ukp
```


# Program CompoundInterest.py: function `table` (2)

- **Problem solving** - Convert the pseudocode below into a sequence of Python statements in your program.
  1. Use `def` to define a function named `table` that has no parameter variable, so the name should just be appended with a pair of brackets `()`.
  2. Read in an initial balance that should be converted to a floating-point value, and store this value in a variable named `initialBalance`.\*
  3. Read in a number of years and store this value in a variable named `numberYears`.
  4. Display the initial balance and the number of years together with some text to specify the meaning of the printout.

Two inputs required.

An example is shown below for the variable `initialBalance`.

```
print("The initial balance is %.2f ukp" % initialBalance)
```

 **Hint \***: Use the function `input` to firstly read in the balance, and then use the function `float` to convert the input string to a floating-point value.

# Program CompoundInterest.py: function table (3)

- **Problem solving** (continued)

Align the print statements (step 4) and the for statement.

5. Write a `for` statement to iterate over a range of values: -6, -3, 0, 3, 6, for the rate. Below is an outline of the algorithm needed to solve the given problem in the `for` loop body.

```
for rate in range( Hint: use -6 as the start value and 3 as the step value. Work out the ending value yourself. ):  
    # Call the function balance below to calculate the  
    # balance for each rate value and store the result in b.  
    b = complete the code to call the function balance together with the arguments: initialBalance, rate and numberYears  
    # Add a print statement below to display a formatted  
    # output of the rate and balance for each rate value.  
    print("define a format string that displays the rate and balance using two suitable format specifiers" % ( specify the two variables, rate and b, here ))
```

The statements have the same indentation.

# Program CompoundInterest.py: function `table` (4)

- **Note**

- The statements defined for steps 2 to 5 have the **same** level of indentation in the function body. Also the two statements in the `for` loop body have the same indentation.
- The string format operator `%` is covered in “Week 5: Strings and Output”.

- Provide a comment to explain the purpose of the function `table`. Include in your comment your name and the date.
- Test the function `table` using suitable values for the initial balance and the number of years. In order to test the function, you need to add a statement that calls the function i.e.,



```
table()
```

The above statement must be defined outside of the function definition for `table`, and it is aligned with the function header.

- Save the function `table` in the file `CompoundInterest.py` and then run your program.



# Supplementary Questions for Private Study

- The laboratory worksheet contains supplementary questions in section 4 for private study.
- You are encouraged to complete the supplementary questions at home, or in the laboratory if you have time after completing questions 2 to 3.