PYTHON BASICS Ebook

www.computervision.zone

Table of contents

Introduction **4**
- Python, are you kidding me? 4
- Installations 4
- Installing Python 5
- Installing IDE 5
- Basic Functions 5
- Indentation 6
- Comments 6

Let the Game Begin **7**
- Numbers and Arithmetics 7
- Strings 7

Data Types **8**
- Basic Variables 8
- Naming Convention 8

Lists **12**

Mutability **14**

Slice that Cake (Slicing) **14**

User Input **15**

Tuples **15**

Sets **16**

Dictionaries **17**

Operators **19**
- Membership Operators 19
- Logical Operators 19
- Comparison Operators 20

IF Statements **21**

Loops **24**
- For Loop 24
- While Loops 26
- Skipping and Breaking Loops 27

Functions **28**

Local and Global Variables 29
Catch that Error (Try/Except) 30
Read and Write Files 31
Classes and Objects 32
Packages and Modules 33
External Packages/Libraries 34
Documentation 34
Key Words 35

Introduction

Python, are you kidding me?

No. Not really. Python is now becoming one of the most popular programming languages. Its branches are now spreading out from simple game development to Automation, Web Development, Data Science and more. With several world class companies such as Google, Facebook, Instagram and Netflix using python, it has opened up numerous job opportunities in various sectors.

Python is a high level general purpose programming language. But what does that mean? An easy way to understand this is to say High level languages are the ones that are closer to human languages therefore easy to understand and maintain. On the other hand low level languages are meant to be understood directly by the machines/hardware, therefore making them difficult to understand. Nowadays the levels have become more of a relative term. For a C programmer Assembly is Low level but for a python programmer even C is a low level programming language. This is not a bad thing, it just means your learning curve to python would be much smaller since you will have a lot of tools at your disposal.

High Level Low Level

Python --- Java / C++ --- C --- Assembly Language --- Machine Code

So the question is, if python is a high level language and machines can only understand Machine code then how does the machine understand Python? The answer to that is the interpreter. If you look up the meaning of interpreter in a dictionary it simply means translator, and that is exactly what is happening here. The interpreter translates the python code to Machine code for the machine to understand. It's pretty much the same as a compiler in other languages with some small differences.

Installations

Python is a programming language and requires a text editor to write and run the code. Therefore we first have to install python and then the text editor or the IDE. IDE stands for Integrated Development Environment. It is a text editor with more tools and functionalities to allow the coding process to be more efficient.
There are many different types of Editors out there one of the most common ones include:
IDLE
Pycharm
Atom
Sublime

Installing Python

First we will start with the python installation. The Installation file can be downloaded from the python website download page:
https://www.python.org/downloads/
For basic learning you can download the latest version. On the other hand for more advanced applications it's better to stick with a stable version rather than the latest one.

Installing IDE

We will be using the Pycharm IDE for this course. If you already have a preferred IDE feel free to continue with that as this course is IDE independent, as long it supports python. There are two editions of Pycharm. We will be using the Community Edition as it is free and is more than enough to get started. Pycharm is available for Windows, Mac and Linux.
To download head on to the download section of the pycharm website and select Community Edition.
https://www.jetbrains.com/pycharm/download/

If you are using Rasbian as your OS i.e you are using Raspberry pi, then you can use the preinstalled Mu IDE.

Basic Functions

To start off with python we will first learn three simple built in functions that will help us throughout this course.

print(): outputs the result on the console
len(): shows the size of the data type
type(): show the type of the data type

Code:

```
print('Hello World')
myData = 'Hello World'
print(len(myData))
print(type(myData))
```

Result:

```
Hello World
11
<class 'str'>
```

Indentation

In most Languages indentation is just for writing a cleaner code but with python the correct indentation is necessary. This means even a wrong space in the beginning of the line can output an indentation error. This might seem a bit annoying in the beginning but as you get used to it , it actually helps in writing cleaner and more readable code.

Correct:

```
print('Hello World')
print('Welcome to the Course')
```

Wrong:

```
print('Hello World')
  print('Welcome to the Course')
```

You can use the tab button to create indentation and you can use the Shit + Tab to remove indentation. This works on single lines as well as multiple lines.

Comments

Comments are used to explain what different lines of your code actually do. They do not get executed with the code, therefore there is no special syntax(rules) that needs to be followed in the comments. To write a comment we simple start it with a hash '#'. The comments can be above a line of code or infront as well.

Code:

```
# This line will print Hello world on the console
print('Hello World')
```

Or

```
print('Hello World')   # This line will print Hello world on the console
```

If you have a line of code that you are not sure about or temporarily want to remove from the code you can comment it out as well. This way it becomes easier to find problems in the code.

Code:

```
#print('Hello World')
```

Let the Game Begin

Numbers and Arithmetics

Lets have a look at some basic Mathematics in python

print(3)	Output: 3	Printing Numerical Value
print(4+5)	Output: 9	Adding Numbers
print(10-5)	Output: 5	Subtracting Numbers
print(3*2)	Output: 6	Multiplying Numbers
print(12/5)	Output: 2.4	Dividing Numbers
print(12//5)	Output: 2	Dividing numbers and removing decimal points.
print(23)**	Output: 8	2 Power 3 resulting in 8
print(6%2)	Output: 0	Modulus results in 0 as no remainder
print(5%2)	Output: 1	Modulus results in 1 as 1 is the remainder

Strings

Strings are simply plain text. So lets see how we can use them.

print('Name')	Output: Name
print('Name = ' + ' Murtaza')	Output: Name = Murtaza (Strings can be added like numbers)
print('ID = ' + 356)	Output: Error - String and numbers cannot be added. Therefore we have to convert the number to string
print('ID = ' + str(356))	Output: ID = 356

Data Types

If we think about programming in a general sense all we are doing is, collecting information, applying some logic and sending an output. The main element where it all begins is the information or the data. We should be able to store this data efficiently, so we can use it for processing and applying logic.The most basic types of data consists of Numbers and Texts. Most of the other data types derive from these two data types.
We will discuss this further but first let's have a look at what variables are.

Basic Variables

Variable is like a container that has something stored inside it. A variable would have a name and a value.

Naming Convention

The naming convention is the style in which the variables,functions and classes etc are declared. They are used for consistency and to provide an easier way to read. This is important when more than one word is used to declare a variable. Let's say we wanted to declare a variable that would store the name of a mobile phone company. If we write the variable as 'phonecompany' it becomes hard to read. There are several types of naming conventions the three most commonly used are the Camel Case, Pascal Case and the Snake Case.
Camel Case: phoneCompany
Pascal Case: PhoneCompany
Snake Case: phone_company

For personal projects and learning it comes down to personal preference. On the other hand when multiple developers are working on a project then a standard convention is followed to avoid any confusions. For this course we will be using the camelCase Convention.

Let's have a look at the basic type of Variables. Say there is a mobile shop and we are developing a program that will help the shopkeeper to do inventory i.e store information of different types of phones their prices and details that the shop has in stock.

For every phone we will need to enter its model number . Now this is a number so our Data Type will be " Integer" . Integers can only store numbers

modelNumber = 309835

Then we will define the screen size. Here the screen size is again a number but this time it has decimal point too. So this time the data type will be 'float'. Floats are numbers with decimals.

screenSize = 5.3

Then we will define the name of the company. This will be plain text so the data type will be ‘String’. String is simply text that can contain alphabets and numbers.

phoneCompany = 'Samsung'

So lets assume this is the only info that is required along with one last piece of information, which is wether this phone is in stock or not. So the data type here will be ‘boolean’ which can have a value of either ‘True’ or ‘False’.

available = True

Note: Even though the data types are all different we don't have to define that in the program. Unlike other languages python automatically detects which data type this variable should be and assigns it. To make sure the data types have been assigned properly we can check them using the ‘type’ function.

Code:

```
print(type(modelNumber))
print(type(screenSize))
print(type(phoneCompany))
print(type(available))
```

Result:

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

So here we can see our data types have been correctly defined starting from integer,float string to boolean.

So why declare Variables ?
The question is that if the variable is just a container that has a value inside it, why not use the value directly, why name it and then use it indirectly? This is becasuse variables can be used in multiple places in the code. So if you change the variable value in the beginning of the code it will change the value everywhere you used that variable. Now this might seem lazy for a few lines of code, but the real advantage becomes clear when we have hundreds of lines of code.

Lets take an example. Say we were writing an article about this phone. We will use the print statement to write up the article. Inorder for us to see better,we will write them in different lines and use the addition property of the strings.

Code:

```
print(  'The company Samsung has released its latest phone with a '+
        'massive 5.3 inch lcd screen. Even though Samsung '+
        ' is not new to large smart phones, the 5.3 inch lcd '+
        'comes as a surprise to most users.')
```

So let's say we declare the variable for Company Name. Now we can replace the text with this variable where ever required.

Code:

```
phoneCompany = 'Samsung'
print(  'The company ' +phoneCompany+' has released its latest phone with a '+
        'massive 5.3 inch lcd screen.Even though '+phoneCompany+
        ' is not new to large smart phones, the 5.3 inch lcd'+
        'comes as a surprise to most users.')
```

Result:

The company Samsung has released its latest phone with a massive 5.3 inch lcd screen. Even though Samsung is not new to large smart phones, the 5.3 inch lcd comes as a surprise to most users.

So now if for some reason we have to go back and change the name of the company or correct the spelling, when we change it in the variable it will automatically correct it in all the places in the article.

Now if we look at the variable screenSize it is a float. And as we have seen before we cannot add numbers to string i.e float + string. So we have to convert float to string before we can add it to the text. We can do this by the 'str()' function .

Converting float to String

Code:

```
screenSizeStr = str(screenSize)
print(type(screenSizeStr))
```

Result:

```
<class 'str'>
```

Lets have a look at some other conversations as well.

Converting float to int using 'int()'

Code:

```
print(screenSize)
print(int(screenSize))
```

Result:

```
5.7
5
```

Converting int to float using 'float()'

Code:

```
print(modelNumber)
```

```
print(float(modelNumber))
```

Result:

```
309835
309835.0
```

Methods

These basic data types have some built in methods.
For example

Code:

```
phoneCompany = 'Samsung'
print(phoneComany.lower())
print(phoneComany.upper())
```

Result:

```
samsung
SAMSUNG
```

The lower method converts all alphabets to lowercase while the upper method converts all alphabets to uppercase.

Lists

Lists are a sequence of basic data types (also known as array). For example if we wanted to store the names of fruits available in a store we could use a list. Lists can be declared with square brackets and inside the bracket we can add items with separation of commas.

myFruits = ['Orange','Apple','Mango']

Accessing the items from a list

The items in a list are ordered and indexed.We can access the items in a list by using their index. The index starts from 0. Therefore in our example Orange would have an index of 0, apple 1 and mango 2.To access we can use square brackets while using the name of the list.

Code:

print(myFruits[1])

Result:

Apple

Adding and Removing Items

We can add items to the list and we can remove items from a list.To add we simply use the append method. Using append adds a value to the end of the list. To remove items we can use the pop method. We can specify an index in the pop method for example if we used index 2 it would remove ‘mango’ from the list.

Code:

```
myFruits.append('Strawberry')
print(myFruits)
myFruits.pop(2)
print(myFruits)
```

Result:

```
['Orange', 'Apple', 'Mango', 'Strawberry']
['Orange', 'Apple', 'Strawberry']
```

Multiple Data Types in a single List

The items in a list don't have to be of the same data type. This means we can have integer float String boolean all in a single list.

Code:

```
phoneDetails = [309835,5.3,'Samsung',True]
print(phoneDetails)
```

Result:

```
[309835, 5.3, 'Samsung', True]
```

Functions with lists

There are several built in functions that work with lists. These functions help us find meaningful information within our lists eg. the total number of items.
Below are a few of the commonly used functions with lists.

len() - returns total number of items present in a list
max() - returns maximum number or highest alphabet
min() - return minimum number of lowest alphabet
sorted() - sorts in ascending order by numbers or alphabets

Code:

```
myFruits = ['Orange','Apple','Mango']
print(len(myFruits))
print(max(myFruits))
print(min(myFruits))
print(sorted(myFruits))
```

Results:

```
3
Orange
Apple
['Apple', 'Mango', 'Orange']
```

Code:

```
myPrices = [23,560,79]
print(len(myPrices))
print(max(myPrices))
print(min(myPrices))
print(sorted(myPrices))
print(sorted(myPrices,reverse=True))
```

Results:

```
3
560
23
[23, 79, 560]
[560, 79, 23]
```

Mutability

Mutability means the ability to change. Some data types are mutable while others are not.For example List is Mutable where strings are not. This means after declaring, the list can be changed but the string cannot be changed.

Changing item of a list

Code:

```
myFruits = ['Orange','Apple','Mango']
myFruits[1] = 'Straberry'
print(myFruits[1])
```

Result:

Straberry

Changing character of a String

Code:

```
fruitOfTheDay = 'Manjo'
fruitOfTheDay[3] ='g'
```

Result:

TypeError: 'str' object does not support item assignment

Slice that Cake (Slicing)

```
myFruits = ['Orange','Apple','Mango','Strawberry','Banana']
print(myFruits[2:4])
['Mango', 'Strawberry']
```

```
myFruits = ['Orange','Apple','Mango','Strawberry','Banana']
print(myFruits[2:])
['Mango', 'Strawberry', 'Banana']
```

myFruits = ['Orange','Apple','Mango','Strawberry','Banana']
print(myFruits[:2])
['Orange', 'Apple']

User Input

We can ask a user to input some information using the input function. The only argument required is the prompt. The prompt is just a message that the user will see, it is not part of the code i.e does not require any syntax.

Code:

```
name = input('Please enter your name: ')
print(" Hi " + name)
```

Result: (Input: Murtaza)

```
Please enter your name: Murtaza
Hi Murtaza
```

Once the user inputs some data we store it in the name variable. Therefore we can now use this variable in our code. We can also request user to input a number but the data received will still be a string. So we can manually change it to a number using the int() or float() functions.

```
pinNumber = int(input('Please enter your pinNumber: '))
```

Tuples

Tuples are similar to lists but they are immutable meaning they cannot be changed after their declaration. Mostly tuple elements are closely related. You can get the items in a tuple but you cannot set or assign a value to a tuple. Tuples can be declared by parentheses with commas or items with only commas.

```
myFruits= ('Orange','Apple','Mango)
```

or

```
myFruits= 'Orange','Apple','Mango'
```

Accessing Tuples (GET)

Code:

```
favFruit = myFruits[2]
print(favFruit)
```

Result:

```
Mango
```

Assigning Tuples (SET)

Code:

```
myFruits[2]= 'Straberry'
```

Result:

```
TypeError: 'tuple' object does not support item assignment
```

Sets

Set is a collection of unique elements. It is Mutable but not ordered. The key property here is the uniqueness. In other words it removes the repetition of the same elements . e.g. [2,2,3,5,5] would become {2,3,5}.
Sets are written with curly brackets e.g.

mySet = {1,2,3,5}

Lets say we gave 20 people 1 question which had 10 options to choose from (1 to 10). Now these answers are stored in a list. If we were interested to know only the answers that were selected we would use set.

Code:

```
surveyAns = [3,6,6,5,8,3,5,1,6,9,5,1,3,6,3,1,5,8,8,9]
optionsSelected=set(surveyAns)
print(optionsSelected)
```

Result:

{1, 3, 5, 6, 8, 9}

To add a new value to the Set we can use .add method. This is different from append since it will add only if the Value is not available already.

Code:

```
mySet = {1,2,3,5}
mySet.add(4)
print(mySet)
```

Result:

{1, 2, 3, 4, 5}

Code:

```
mySet.add(2)
print(mySet)
```

Result:

{1, 2, 3, 4, 5}

Since the value 2 is already in the set therefore the set remains unchanged

Dictionaries

Where a list is a powerful data type which allows sequential storing, it lacks the element of dimensionality. A list has only one dimension. Therefore storing 2 dimentinal information in a list is not very feasible. For example the names of the phones and thrier prices. With dictionaries you can store values like in a table . Dictionaries are Mutable and unordered, therfore they can be changed after the declaration as well .

There are two Elements of a Dictionary.

1. Key
2. Value

Dictionaries can be written using curly brakets with the key and value seperated by colon and the items sepreated by commas

myDictionary = {'key1':value1,'key2':value2,'key3':value3}

So it is like an actual dictionary where every word is the key and its meaning would be the values. Let's take the example of the mobile shop . Here we want to store the names of each phone and its price.

Code:

```
priceList = {'iphone8':600,
             'iphone9':700,
             'iphoneX':800,
             'ihone11':1000}
```

Accessing the value of a key

We can access the value of a key by using square brackets along with the key itself.

Code:

```
iphone8Price = priceList['iphone8']
print(iphone8Price)
```

Result:

600

Assigning the values

Dictionary Values are Mutable that means if the value of the iphoe11 decreases we can change it in the dictionary.

Code:

```
priceList['iphone11'] = 900
print(priceList['iphone11'])
```

Result:

900

Note:

Keys of Dictionary are Immutable

Values of Dictionary are Mutable

Nested Dictionaries

Coming back to our mobile shop example, the shop keeper would want to store details of each phone but not just with its price but with other features such as company name, screen size, model number etc. In this case we can use nested Dictionaries, which simply means a dictionary inside of a dictionary.

Let's take an example, here we will store the iphone type and then its two features price and size.

Code

```
iphoneDetails = {'iphone8':{'price':600,'size':4.7},
                 'iphone9':{'price':700,'size':4.7},
                 'iphoneX':{'price':800,'size':5.8},
                 'Iphone11':{'price':1000,'size':6.1}}
```

Accessing the values

The values can be accessed as before. This time it will just contain another bracket with with the second key.

Code:

```
iphone8Price = iphoneDetails['iphone8']['price']
iphone8Size = iphoneDetails['iphone8']['size']
print(iphone8Price,iphone8Size)
```

Result:

```
600 4.7
```

Assigning the values

Code:

```
iphoneDetails['iphone11']['price'] = 900
print(iphoneDetails['iphone11'])
```

Result:

```
{'price': 900, 'size': 6.1}
```

Operators

Operators help us perform operations on Variables and values.

Membership Operators

Operator	Description	Example
in	Checks if an item or value is present in the given data	**'Apple' in myFruits** True is apple present False if not present
not in	Checks if an item or values is not in the given data	**'Apple' in myFruits** True is apple present False if not present

Logical Operators

Operator	Description	Example
and	Checks both sides and compares if both are true or not	**A and B** True: if both are ture False: if one of them is False
or	Check if one of the two sides is true	**A or B** True: one of them is True Flase: Both of them are Flase
not	Reverse the value	**not(A)** True: if A = False False: if A = True

Comparison Operators

Operator	Description	Example
>	Checks if the value on the left is greater than the value on the right	3 > 5 Result: False 5 > 3 Result: True
<	Checks if the value on the left is less than the value on the right	3 < 5 Result: True 5 < 3 Result: False
==	Checks if both value are equal	3 == 3 Result: True 3 == 5 Result: False
>=	Checks if the value on the left is greater than or equal to the value on the right	2 >=2 Result: True 3 >=2 Result: True 2 >=3 Result: False
<=	Checks if the value on the left is less than or equal to the value on the right	5 <=5 Result: True 3 <=5 Result: True 5 <=3 Result: False
!=	Check if the values are not equal	3 != 5 Result: True 3 != 3 Result: False

IF Statements

The if statement allows to check a condition and execute a block of code if its true and execute some other block of code if its not true. Let's say we had a list of items in the store and we wanted to write a program to check if an item is available or not . We will check the list, if the item is available we will return available else we will return not available.

Code:

```
stock = ['Keyboard','Mouse','Headphones','Monitor']
productName = input('Which product would you like to look up: ')
if productName in stock:
   print('Available')
else:
   print('Not Available')
```

Input:

Mouse

Result:

Available

Input:

MousePad

Result:

Not Available

Here ‘stock’ is our list that contains 4 items of Strings. We are asking the user to input a string and then we store it in the variable productName . Now we can use our membership operator to check whether the productName received from the user e.g. ‘Mouse’ is in the stock list or not. So if it is in the list it will return the value True, therefore running our first block of code and printing ‘Available’. In the second instance the product name ‘MousePad’ was not in the stock list therefore returning False and getting in the else block of code and returning ‘Not Available’.

Complex Conditions

We can also check multiple conditions and perform an action based on their output.
To demonstrate this lets create a Guessing Game. We will ask the user to guess a number from 0 to 10 . If the number is correct we will output correct answer, if wrong we will output wrong answer. Now to generate a random number every time we are going to use a module by the name random . But what is a moudle? Well we will dicsus this futher but for now all you need to kown is that we are using a function form this module that will help us generate random numbers.

Code:

```
import random
number= random.randint(0,11)
print('Number to Guess:',number)
myNumber = int(input('Guess the number: '))

if myNumber < number or myNumber >number:
   print('Your Guess is Wrong')
else:
   print('Your Guess is Correct')
```

Result:(Input: 3)

```
Number to Guess: 5
Guess the number: 3
Your Guess is Wrong
```

Result:(Input: 2)

```
Number to Guess: 2
Guess the number: 2
Your Guess is Correct
```

If more than one condition is to be checked than the else if statement can be used with the if statement.

Code:

```
import random
number= random.randint(0,11)
print('Number to Guess:',number)
myNumber = int(input('Guess the number: '))

if myNumber < number:
   print('Your Guess was lower than the Value')
elif myNumber > number:
   print('Your Guess was higher than the Value')
else:
   print('Your Guess is Correct')
```

Result:(Input: 4)

```
Number to Guess: 7
Guess the number: 4
Your Guess was lower than the Value
```

Result: (Input: 7)

```
Number to Guess: 4
Guess the number: 7
Your Guess was higher than the Value
```

Result:(Input: 6)

Number to Guess: 6

Guess the number: 6

Your Guess is Correct

Loops

Loops run a block of code over and over again based on a condition or count. There are two main types of Loops, ‘While’ and ‘For’ Loops. In other words we can access one element at a time from Lists, Dictionaries,Tuples etc

For Loop

The for loop runs for a specified number of time.There are two main ways you can use the for loop .

1. Using Fixed number
2. Using Items

Using Fixed Number: we can define the number of time the loop should repeat, using ‘in range’.

Code:

```
for x in range (0,3):
   print(x)
```

Result:

```
0
1
2
```

Using Items: we can loop based on the number of items present in list, dictionary, tuple etc.

Code:

```
myFruits = ['Orange','Apple','Mango']
for item in myFruits:
   print(item)
```

Result:

```
Orange
Apple
Mango
```

Lets say we wanted to make all our fruit name plural . So we would iterate through our list and add ‘s’ to each element

Code:

```
myFruits = ['Orange','Apple','Mango']
for x in range(len(myFruits)):
   myFruits[x] =  myFruits[x] + 's'
print(myFruits)
```

Result:

```
['Oranges', 'Apples', 'Mangos']
```

For loop with Dictionaries

While using for loops with dictionaries we can unpack the key along with the values. To achieve that the .items() methods has to be used.

Code:

```
myFruits = {'Orange':25,'Apple':56,'Mango':65}
for key,val in myFruits.items():
    print(key,val)
```

Result:

```
Orange 25
Apple 56
Mango 65
```

While Loops

‘While’’ loops run for an indefinite number of iterations which means it could run forever if the condition is met. Lets build a simple counter that would count down to start a game.

Code:

```
from time import sleep
countDown= 5
while (countDown != 0):
   print(countDown)
   sleep(1)
   countDown -= 1
print('Let the game begin')
```

Result:

```
5
4
3
2
1
Let the game begin
```

Skipping and Breaking Loops

Skipping an iteration of the loop can be helpful in many cases. Let's have a look at how it can be done. We are going to build a program where we can find all the odd numbers from a given list.

Using ‘continue’

Code:

```
numList = [1,2,3,4,5,6,7,8,9,10]
oddNumList =[]
for num in numList:
   if (num % 2 == 0):
     continue
   oddNumList.append(num)
print(oddNumList)
```

Result:

```
[1, 3, 5, 7, 9]
```

The break statment can be used to get out of the loop at any given time. Lets look at an example. Here we will create a loading bar which will show the percentage of laoding complete. Once complete it will break the loop .

Using ‘break’

Code:

```
from time import sleep
per= 0
while True:
   if per==100:
      break
   print(per)
   per += 20
   sleep(1)
print('Loading Complete')
```

Result

```
0
20
40
60
80
Loading Complete
```

Functions

Function is simply a named block of code in your program. This can be called to run at different times with different parameters. The main advantage of a function is that it can be resued i.e it can be called multiple times.Function allow better organization of the code. We will write a function to find the circumference of a circle of known radius.

Code:

```
def cir(radius):
   c= 2*3.14*radius
   return c
myCir = cir(2)
print(myCir)
```

Result:

```
12.56
```

To really understand the advantage of a function we will use multiple values of radius to get multiple values of the circumference. This way we call the function several times rather than hard coding.

Code:

```
myRadii= [1,2,3,4,5,6,7,8,9,10]
myCir= []
def cir(radius):
   c= 2*3.14*radius
   return c

for r in myRadii:
   myCir.append(int(cir(r)))

print(myCir)
```

Result:

```
[6, 12, 18, 25, 31, 37, 43, 50, 56, 62]
```

Local and Global Variables

Local variables can only be used in a particular region e.g. in a function . Whereas the global variables can be used anywhere in the code.

Lets look at an example of a local variable. If we define pi in our function it becomes a local variable and therefore cannot be accessed outside the function.

Code:

```
def cir(radius):
    pi = 3.14
    c = 2*pi*radius
    return c
print(pi)
```

Result:

```
NameError: name 'pi' is not defined
```

If we declare the pi variable outside the fucntion then it become global variable . Therefore it can now we access both in the fuction and outside the fuction

Code:

```
pi = 3.14
def cir(radius):
    c = 2*pi*radius
    print(pi)
    return c
cir(3)
print(pi)
```

Result:

```
3.14
3.14
```

Catch that Error (Try/Except)

Let's say you are logging in to your bank's website and they ask you to enter your atm pin number for verification. The code for this is quite straightforward.To make sure we get a pin we can convert the string to an integer.
Code:

```
atmPin = int(input('Please Enter your 4 digit Pin '))
```

Now for God knows what reason, if you decide to enter alphabets instead of digits the program will run into an error.
Result: (input: abcd)

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

This will cause the whole program to crash. So that is not very practical, therefore we need to catch this error when it happens. For this we have the try and except.

Code:

```
try:
   atmPin = int(input('Please Enter your 4 digit Pin '))
   print('Input Accepted')
except:
   print('Input Invalid')
```

Result:

(Input: 1234)
Input Accepted

(Input: abcd)
Input Invalid

Read and Write Files

Reading and writing files uses the open() function . Here we have to define the file name and the type of permission. Below are examples of how to Read, Write and Append files. The difference between appending and writing is that appending adds to the existing data whereas writing overwrites the data.

Writing to a file

```
f= open("myFile.txt","w+")
for x in range(0,10):
    f.write('Current Line= ' +str(x)+'\n')
f.close()
```

Append Files

```
f= open("myFile.txt","a+")
for x in range(0,10):
    f.write('Current Line= ' +str(x)+'\n')
f.close()
```

Read File

```
f= open("myFile.txt","r+")
myData = f.read()
print(myData)
```

Classes and Objects

So far we have looked at built in data type such as int float , string lists , dictionaries etc. Now sometimes these data types are not enough to efficiently handle the information. Let's say we want to define a human. So the human would have attributes such as name, height, age, history of job, spouse, phone number etc. So all of this cannot be but in a single data type. So we can create our own data type and call it human. So this new data type will be a class. And when we create a new human and define its parameters that will be an object. So Class is the template and object is a sample created using this template. In Other words, an object is an instance of a class.
As a matter of fact all data types in python are already classes. If we use the type function to check any data type it would say class and then the name of the data type.

So let's create our human data type by defining a class .

Code: (file 1)

```
class Human:
    def __init__(self,name,age,height):
        self.name = name
        self.age = age
        self.height = height

    def heightCm(self):
        return self.height*30.48
```

Code:(file 2)

```
from HumanClass import Human
human1 = Human('Murtaza',35,6)
print(human1.age)
print(human1.heightCm())
```

Packages and Modules

Modules are simply python files that contain classes, functions and variables. So they can be thought of as helping tools that you can access to avoid writing everything from scratch. So what are packages then ? Packages are just a bunch of modules put together. They are pretty much similar to libraries. In fact a lot of the times these names are interchangeable. A Package can also contain subpackages which in turn contain the modules. Python has a number of built in packages and modules. Let's have a look at the built-in time module.

To import a module or package we can simply type import and then the name

import time

Now we can use the functions and variables in the time module by simply writing its name

time.sleep(1)

Here we have used the sleep function from the time module.

The above method imports the complete module if we wanted to import just a specific function form the module we can use ‘from’

from time import sleep

Now we can simply use the sleep function by its name

sleep(1)

The packages and subpackages are imported in the same way.
Let's say we created a folder in our main directory by the name Resources and we created another folder in this folder by the name Docs and we created a python file by the name utlis in this folder.
We can then import our python file using the following command.

import Resources.Docs.utlis

These modules or python files can be very useful since you don't want to put hundreds of lines of code in a single file. What we can do is to put our helping functions in a separate file e.g. by the name utlis.py. This would be our Utilities module where we would define all our helping functions,variables and classes. Then we could simply import this module and use its functions.

Import utlis
utlis.calculateArea(20,20)

External Packages/Libraries

Even though python has a lot of inbuilt modules and packages, sometimes they are not enough to get the job done. Therefore requiring external packages. With Pycharm the importing process is very easy . All you have to do is to go to settings in windows or preferences in mac and then In the Project Interpreter you can search and add for different packages.
Importing packages is the same as before. For example we can import the famous computer vision package opencv by:

import cv2

Sometimes the name of the package is long and not convenient to use. In that case you can assign it a name. For example the matplot library allows easy plotting of graphs in python. Instead of using its full name we can import it as plt.

import matplotlib as plt

If you are not using pycharm and would like to install a package you can use pip. Pip comes with python which is a package manager allowing easy installations. For example if we wanted to install opencv we could use the following command in Command prompt in windows and Terminal in mac:

pip3 install opencv-python

Here opencv-python is the name of the package. Similarly you can uninstall packages using pip as well.

pip3 uninstall opencv-python

Documentation

Sometimes writting one line of comment is not enough. In these case you can use the multiple line comment option. This has become a standard for explaining functions, An example is shown below.

Code:

```
pi = 3.14
def cir(radius):
    """
    Calculates the circumference of a circle where the input is the
    radius of the circle.
    """
    c = 2*pi*radius
    print(pi)
    return c
cir(3)
print(pi)
```

Key Words

Compiler: Translator that converts High level code to low level code.
High Level Language: Closer to human language rather than machine code.
IDE: Integrated Development Environment. Text Editor with more features.
Interpreter: Translator that converts High level code to low level code.
Iterable: Iterate through elements . Access one element at a time.
Low Level Language: Closer to machine code rather than human language
OOP: Object Oriented Programming
Packages: A collection of Modules
Parenthesis: ()
PIP: Package Manager, helps in installation of packages and modules
Syntax: rules of a programming language
Variable: Container that holds a value