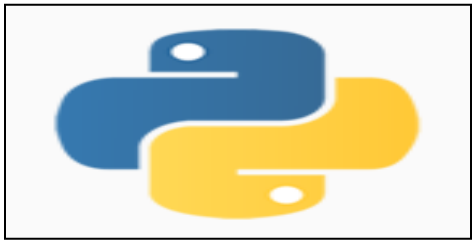


Python

Dr. Shahid Mahmood Awan
Assistant Professor
School of Systems and Technology, University of Management and Technology
shahid.awan@umt.edu.pk

Umer Saeed(MS Data Science, BSc Telecommunication Engineering)
Sr. RF Optimization & Planning Engineer
f2017313017@umt.edu.pk

Python

Python Overview

Python Overview

- Python is a general-purpose object-oriented programming language with high-level programming capabilities.

- It has become famous because of its clear and easily understandable syntax, portability and easy to learn.

History

- Python was developed in the late eighties i.e. late 1980's by ***Guido van Rossum*** at the ***National Research Institute for Mathematics and Computer Science*** in the Netherlands as a successor of ABC language capable of exception handling and interfacing.

- Python is derived from programming languages such as ABC, Modula 3, small talk, Algol-68.

- Python page is a file with a **.py** extension that contains could be the combination of HTML Tags and Python scripts.

- In December 1989 the creator developed the 1st python interpreter as a **hobby** and then on **16 October 2000, Python 2.0** was released with many new features.

- On **3rd December 2008, Python 3.0** was released with more testing and includes new features.

History

- Python is an **open source** scripting language.
- Python is **free to download** and use.
- Python is one of the official languages at Google.

Characteristics and Features of Python

- **1- Interpreted Language:** Python is processed at runtime by Python Interpreter.
- **2- Object-Oriented Language:** It supports object-oriented features and techniques of programming.
- **3- Interactive Programming Language:** Users can interact with the python interpreter directly for writing programs.
- **4- Easy language:** Python is easy to learn language especially for beginners.
- **5- Straightforward Syntax:** The formation of python syntax is simple and straightforward which also makes it popular.
- **6- Easy to read:** Python source-code is clearly defined and visible to the eyes.

Characteristics and Features of Python

- **7- Portable:** Python codes can be run on a wide variety of hardware platforms having the same interface.
- **8- Extendable:** Users can add low level-modules to Python interpreter.
- **9- Scalable:** Python provides an improved structure for supporting large programs then shell-scripts.

Installing Python on Windows

Installing Python on Windows

- Windows doesn't come with Python installed.
- You can get a basic copy of Python from the anaconda site; https://www.anaconda.com/download/.
- The following procedure should work fine on any Windows system, whether you use the 32-bit or the 64-bit version of Anaconda.

Installing Python on Windows

- **Locate the downloaded copy of Anaconda on your system:**

- The name of this file varies, but normally it appears as Anaconda3-5.0. 1-Windows-x86.exe for 32-bit systems and Anaconda3-5.0.1-Windows-x86_64 exe for 64-bit systems.

- The version number is embedded as part of the filename.

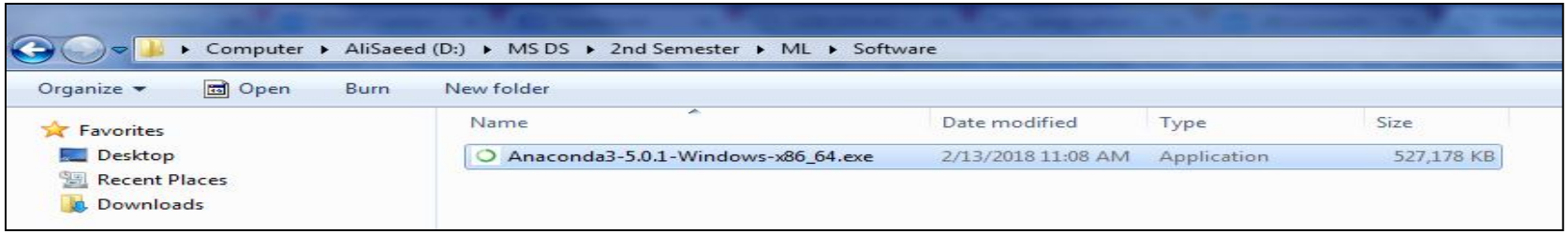
- In this case, the filename refers to version 3-5.0. 1, which is the version used for this presentation.

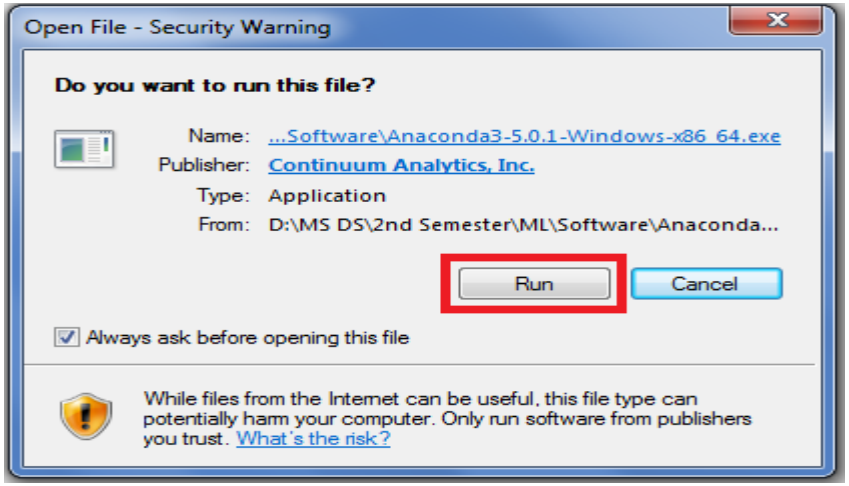
- If you use some other version, you may experience problems with the source code and need to make adjustments when working with it.

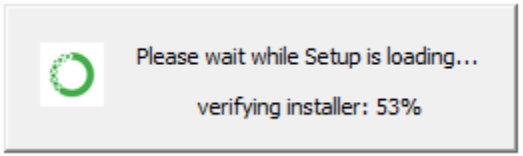
Installing Python on Windows

- **Double-click the installation file**

- You may see an Open File – Security Warning dialog box that asks whether you want to run this file. Click Run if you see this dialog box pop up;







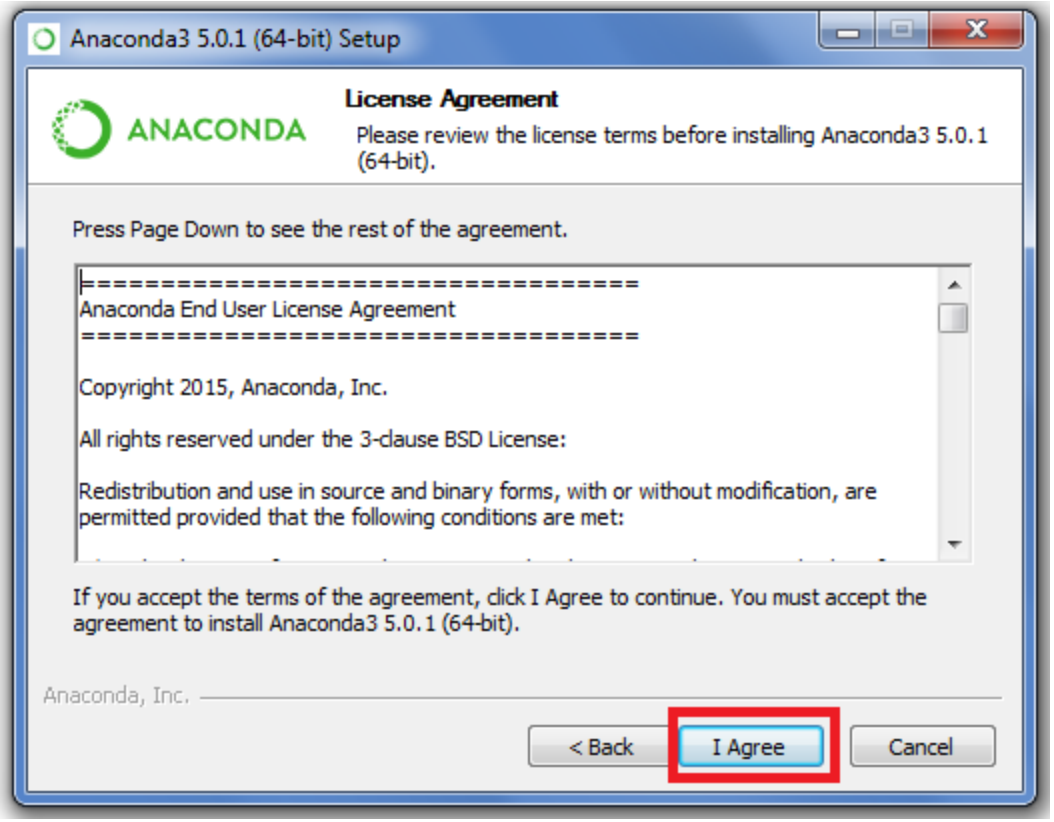
Installing Python on Windows

- You see an Anaconda 3-5.0.1 Setup dialog box similar to the one shown in Figure.
- The exact dialog box that you see depends on which version of the Anaconda installation program you download.
- If you have a 64-bit operating system, using the 64-bit version of Anaconda is always best so that you obtain the best possible performance.
- This first dialog box tells you when you have the 64-bit version of the product.

- Click on “Next”

Installing Python on Windows

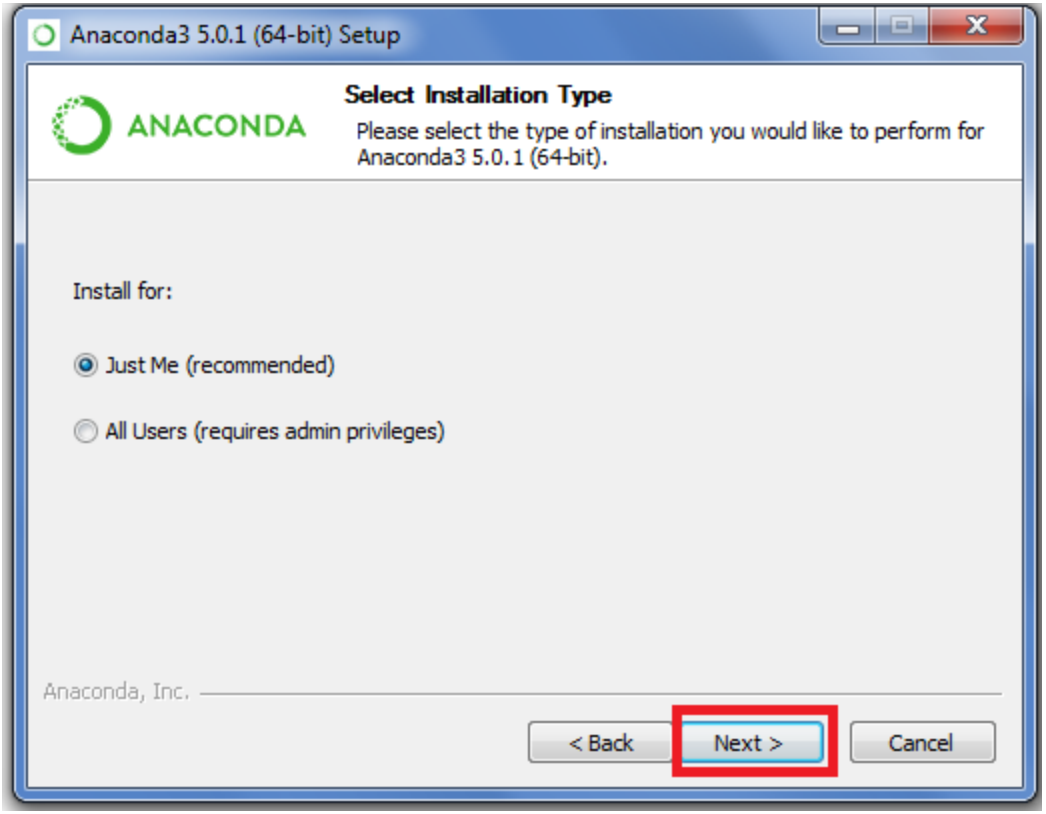
- The wizard displays a licensing agreement. Be sure to read through the licensing agreement so that you know the terms of usage.



- Click on “I Agree” if you agree to the licensing agreement

Installing Python on Windows

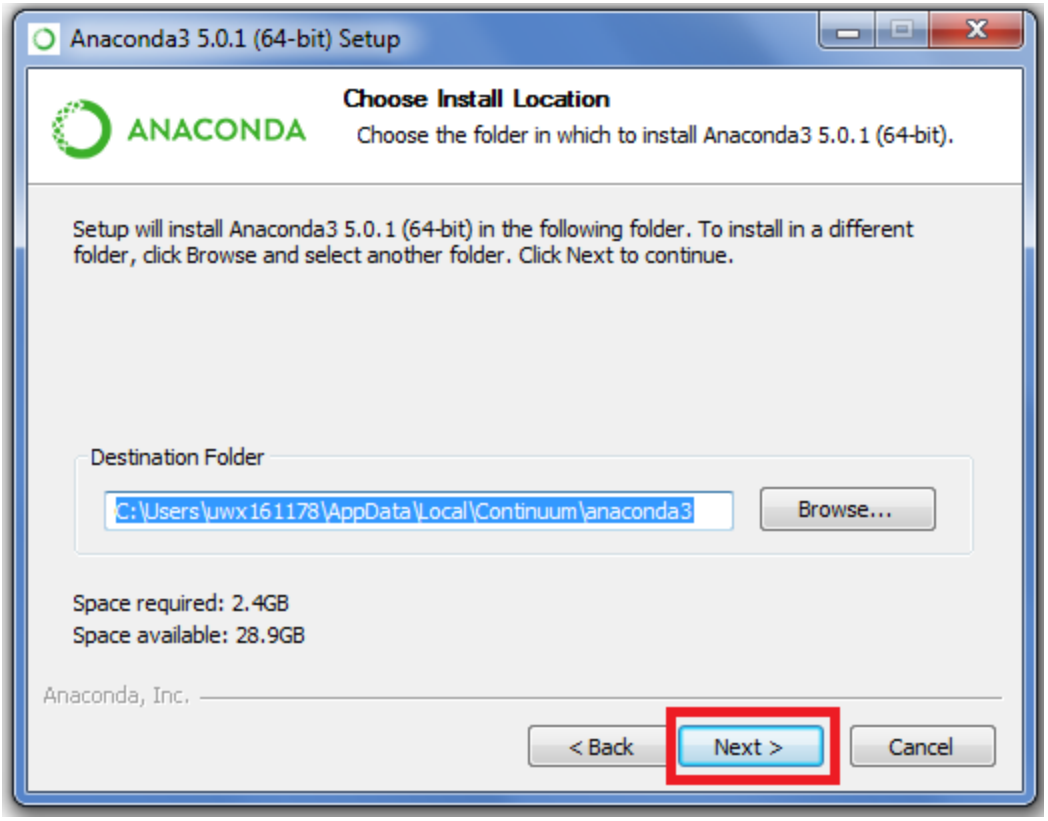
- You're asked what sort of installation type to perform, as shown in Figure. In most cases, you want to install the product just for yourself. The exception is if you have multiple people using your system and they all need access to Anaconda.



- Choose one of the installation types and then click Next

Installing Python on Windows

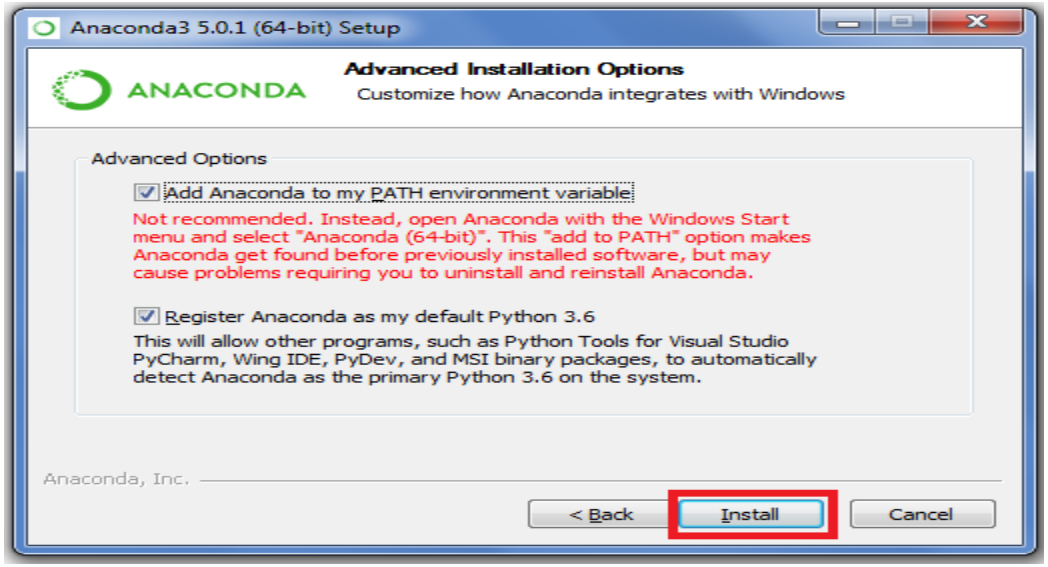
- The wizard asks where to install Anaconda on disk, as shown in Figure. The presentation assumes that you use the default location.



- Choose an installation location (if necessary) and then click Next

Installing Python on Windows

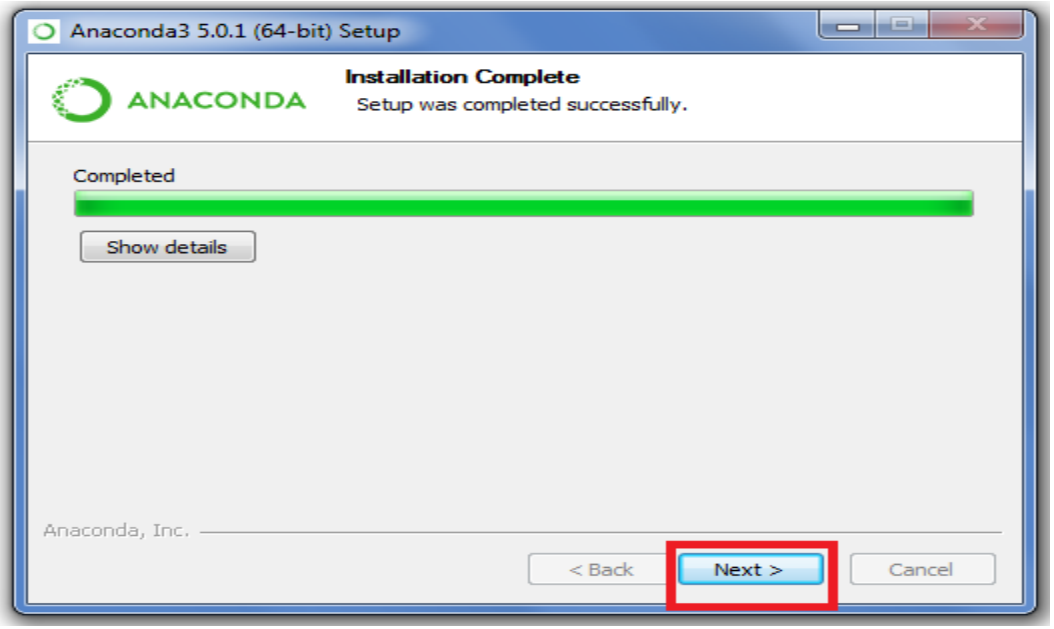
- You see the Advanced Installation Options, shown in Figure .
- These options are selected by default, and no good reason exists to change them in most cases.
- You might need to change them if Anaconda won't provide your default Python 2.7 (or Python 3.5) setup



- Change the advanced installation options (if necessary) and then click Install.

Installing Python on Windows

- You see an Installing dialog box with a progress bar.
- The installation process can take a few minutes, so get yourself a cup of coffee and read the comics for a while.
- When the installation process is over, you see a Next button enabled

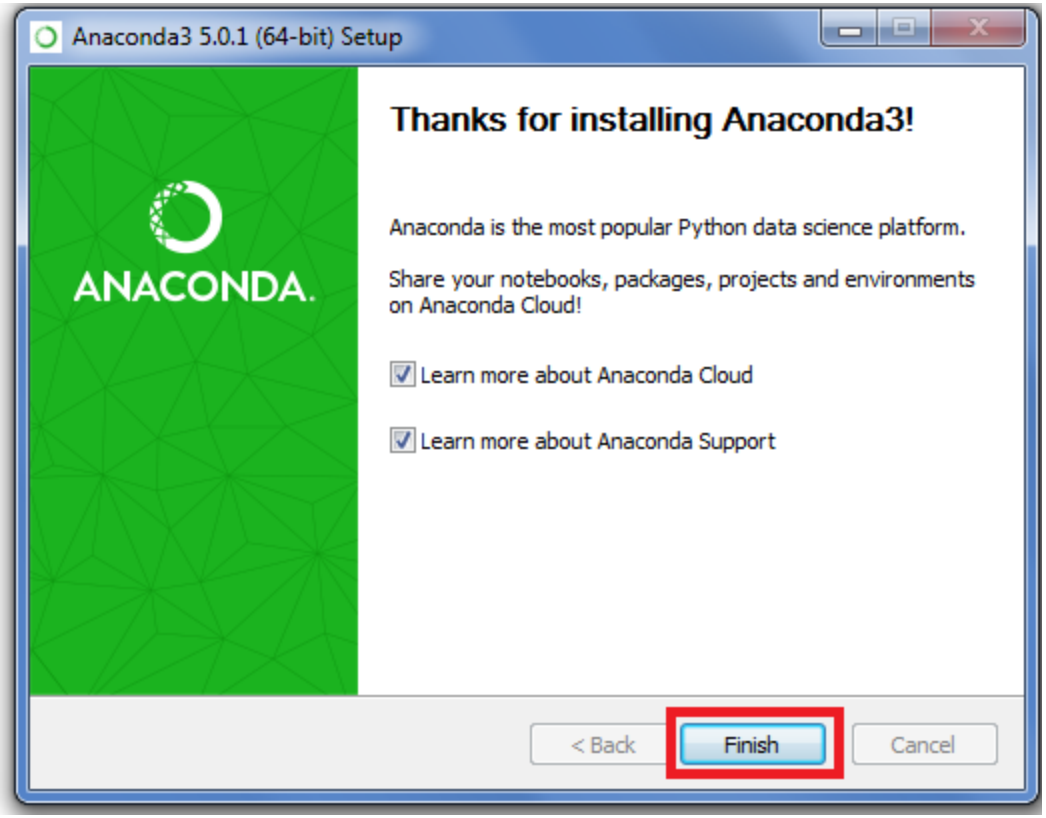


- Click on “Next”

Installing Python on Windows

- The wizard tells you that the installation is complete.

- You're ready to begin using Anaconda.



- Click on "Finish"

Installing Python on Windows

- There are various GUI based Python IDE that python programmers can use for better coding experience.

- Names of some Python interpreters are;
  1. PyCharm
  2. Python IDLE
  3. The Python Bundle
  4. pyGUI
  5. Sublime Text etc.

Variables, Expressions and Statements

Values

- A value is one of the basic parts of a program like a letter or a number.
- These values belong to different *types*: for example, integers and strings, float, complex and logical.
- If you are not sure what type a value has, the interpreter can tell you
- Python Values Examples are;

Value	Statement	Data Type
59	type(59)	integers
Hello, Word!	type("Hello, Word!")	string (combination of letters)
3.2	type(3.2)	float
“59”	type(“59”)	string
“3.2”	type(“3.2”)	String
’59’	type(‘59’)	string
‘3.2’	type(‘3.2’)	String
True	type(True)	Bool
False	Type(False)	Bool

Print

Statement	Ouput
print("Hello Word!")	Hello Word!
print(4)	4

- When you type a large integer, you might be tempted to use commas between groups of three digits, as in 1,000,000. This is not a legal integer in Python, but it is legal:

Statement	Ouput
print(1,000,000)	1 0 0

- Python interprets 1,000,000 as a comma separated sequence of integers, which it prints with spaces between.
- This is the first example of semantic error.
- **semantic error:** the code runs without producing an error message, but it doesn't do the "right" thing.

Python Variables

- One of the most powerful features of a programming language is the ability to manipulate *variables*.
- A variable is a name that refers to a value.
- Variables are identifiers of a physical memory location, which is used to hold values temporarily during program execution.
- Python interpreter allocates memory based on the values data type of variable, different data types like integers, decimals, characters etc. can be stored in these variables.

Python Variables

- An *assignment statement* creates new variables and gives them values:

Assignment Statement	Print Function	Output
message = ”Hello, I am Umer Saeed”	print(message)	Hello, I am Umer Saeed
n = 17	print(n)	17
pi = 3.1415926535897931	print(pi)	3.1415926535897931

- The first assigns a string to a new variable named message; the second assigns the integer 17 to n; the third assigns the (approximate) value of to pi.

Print Function	Output
type(message)	str
type(n)	int
type(pi)	float

Variables Names

- Programmers generally choose names for their variables that are meaningful and document what the variable is used for.
- Variable names can be arbitrarily long.
- Variable names are case-sensitive.
- It is legal to use uppercase letters, but it is a good idea to begin variable names with a uppercase letter.
- A variable name can only contain alphanumeric characters and underscore such as (a-z, A-Z,0-9 and _).
- Variable names must begin with a letter or underscore. Variable names can start with an underscore character, but we generally avoid doing this unless we are writing library code for others to use.
- A variable name can not contents space.

Variables Names

- They can contain both letters and numbers, but they cannot start with a number.
- Reserved words cannot be used as variable name.
- If you give a variable an illegal name, you get a syntax error:

Variable Name	Validity	Reason	Statement
var_name%	Invalid	Has the character '%'. Only underscore(_) allowed.	var_name%="hello"
var.name	Invalid	Has the character '.'. Only underscore(_) allowed.	var.name="hello"
var_name$	Invalid	Has the character '$'. Only underscore(_) allowed.	var_name$="hello"
2var_name	Invalid	Starts with a number is not allowed.	2var_name="hello"
_var_name	valid	Starts with _ which is valid, however not recommended	_var_name="hello"
var name	Invalid	Space is not allowed in variable name	var name="hello"

Python Reserved Keywords List

- The Python Keywords must be in your information because you can not use them as a variable name or any other identifier name.
- Keywords are reserved words in Python and used to perform an internal operation.
- All the keywords of Python contain lower-case letters only (except True and False).

Python Reserved Keywords List					
and	assert	in	as	break	return
del	else	raise	elif	except	def
from	if	continue	global	import	for
not	pass	finally	or	print	lambda
while	yield	is	with	class	try
exec	False	True			

Choosing mnemonic variable names

- As long as you follow the simple rules of variable naming, and avoid reserved words, you have a lot of choice when you name your variables.
- In the beginning, this choice can be confusing both when you read a program and when you write your own programs.
- For example, the following three programs are identical in terms of what they accomplish, but very different when you read them and try to understand them.

Statement 1	Statement 2	Statement 3	Statement 4	Ouput
a = 35.0	b = 12.50	c = a * b	print(c)	437.5
hours=35.0	rate=12.50	pay=hours*rate	print(pay)	437.5
x1q3z9ahd = 35.0	x1q3z9afd = 12.50	x1q3p9afd = x1q3z9ahd * x1q3z9afd	print(x1q3p9afd)	437.5

Choosing mnemonic variable names

- The Python interpreter sees all three of these programs as *exactly the same* but humans see and understand these programs quite differently.
- Humans will most quickly understand the *intent* of the second program because the programmer has chosen variable names that reflect their intent regarding what data will be stored in each variable.
- We call these wisely chosen variable names “mnemonic variable names”.
- The word mnemonic means "memory aid“
- We choose mnemonic variable names to help us remember why we created the variable in the first place.

Assigning Values to Variables

- Python interpreter is able to determine that what type of data is stored, so before assigning a value, variables do not need to be declared.
- Usually in all programming languages, equal sign "=" is used to assign values to a variable.
- It assigns the values of right side operand to left side operand.
- The left side operand of = operator is the name of a variable, and right side operand is value.
- **Example**
- name = "Packing box" # A string
- height = 10 # An integer assignment
- width = 20.5 # A floating point

Assigning Values to Variables

- In the above code, the variable name 'height' is storing a value 10 and since the value is of type integer, the variable is automatically assigned the type integer.
- Another variable name 'width' is assigned with floating type value.
- **Multiple Assignments:**

Statements	Ouput
a, b, c = 1, 2, "Hey"	
print(a)	1
print(b)	2
print(c)	Hey

- You can output multiple values with the print() function. For example, you can do this:

Statements	Ouput
planet = "Earth"	
country = "Pakistan"	
print(planet, country)	Earth Pakistan

Assigning Values to Variables

- You can also combine the variables with other text.

Statements	Ouput
name = "Umer Saeed"	
country = "Pakistan"	
print("My name is " + name + " and I live on planet " + country)	My name is Umer Saeed and I live on planet Pakistan

Python Numbers

Statements	Output
print(type(1))	int
print(type(-1))	int
print(type(1.0))	float
print(type(-1.0))	float
print(type(2e3))	float
print(type(-2e3))	float
print(type(3.14j))	complex
print(type(-3.14j))	complex

- Refers to an ***integer***. An integer is a whole number (i.e. not a fraction). Integers can be a positive number, a negative one, or zero. Examples of integers: -3, -2, -1, 0, 1, 2, 3
- Refers to a ***floating point number***. Floating point numbers represent real numbers and are written with a decimal point dividing the integer and the fractional parts. Floating point numbers can also be in scientific notation, with *E* or *e* indicating the power of 10 (eg, +1e3 is equivalent to 1000.0). Examples of floats: 1.0, 12.45, -10.0, -20.76789, 64.2e18.
- A *complex number* takes the form a + bj where a is a real number and b is an imaginary number. Each argument an be any numeric type (including complex). The first argument can also be a string (but the second argument can't). Examples: 1.4j, -1.4j.

Conversion Functions

- You can also use functions such as int(), float(), and complex() to make an explicit conversion between one number type and another.

Statements	Output
a = 100	
print(type(a))	
print(type(float(a)))	float
print(type(complex(a)))	complex
print(type(bool(a)))	bool

Numbering Systems

- The decimal numbering system is the most widely used system in the modern world. Also called *base-ten*, the *decimal* system has 10 as its base, and uses the digits 0 to 9.
- There are other numbering systems though, that don't use 10 as its base. The *binary* system is base-two (uses the digits 1 and 0),
- the *octal* system is base-eight (uses digits 0 to 7),
- and the *hexadecimal* system is base-sixteen (uses digits 0 to 9, and letters A to F).
- These numbering systems tend to be more popular in mathematics and computing.

Numbering Systems

- In Python, you can specify the numbering system a number uses by using a two-digit prefix as follows:

Numbering System	Prefix
Binary	0b or 0B
Octal	0o or 0O
Hexadecimal	0x or 0X

- The prefix can be uppercase or lowercase.

Statements	Output
print("...", 0b001, 0b010, 0b011, 0b100, 0b101, "...")	……1 2 3 4 5 ...
print("...", 0o05, 0o06, 0o07, 0o10, 0o11, 0o12, 0o13, 0o14, "...")	……5 6 7 8 9 10 11 12 13 ...
print("...", 0x7, 0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf, 0x10, 0x12, 0x13, "...")	…7 8 9 10 11 12 13 14 15 16 18 19 ...

Numbering Systems

- You can also use functions such as hex() and oct() to return an integer as a hexadecimal or octal number.

Statements	Output
a = 100	
print(oct(a))	0o144
print(hex(a))	0x64
print(bin(a))	0b1100100

Operators and Operands

- Python operators are symbols that are used to perform mathematical or logical manipulations.
- Operands are the values or variables with which the operator is applied to, and values of operands can manipulate by using the operators.
- **Example**
- 6 + 2=8, where there are two operands and a plus (+) operator, and the result turns 8.
- Here a single operator is used to manipulate the values.
- The +, -, *, / and ** does addition, subtraction, multiplication, division & exponentiation respectively.

Python Expressions

- An *expression* is a combination of values, variables, and operators.

Statement	Print Function	Output
n = 17	print(n)	17
pi = 3.1415926535897931	print(pi)	3.1415926535897931
o=n+pi	print(o)	20.14159265

Types of Python Operators

- Python programming language is rich with built-in operators.

- The following types of operators are supported by Python:
  1. Arithmetic Operators
  2. Assignment Operators
  3. Comparison (Relational) Operators
  4. Logical Operators
  5. Identity Operators
  6. Bitwise Operators
  7. Membership Operators

Order of Operations

- When more than one operator appears in an expression, the order of evaluation depends on the *rules of precedence*.
- For mathematical operators, Python follows mathematical convention.
- The acronym *PEMDAS* is a useful way to remember the rules:
- ***P*arentheses** have the highest precedence and can be used to force an expression to evaluate in the order you want.

Statement	Ouput
2 * (3-1)	4
(1+1)**(5-2)	8

- You can also use parentheses to make an expression easier to read, as in (min * 100) / 60, even if it doesn't change the result.

Order of Operations

- ***E*xponentiation** has the next highest precedence,

Statement	Ouput
2 * (3-1)	4
3*1**3	3

- ***M*ultiplication and *D*ivision** have the same precedence, which is higher than *A*ddition and *S*ubtraction, which also have the same precedence.

Statement	Ouput
2*3-1	5
6+4/2	8

- Operators with the same precedence are evaluated from left to right. Expression 5-3-1 is 1, not 3, because the 5-3 happens first and then 1 is subtracted from 2.

- When in doubt, always put parentheses in your expressions to make sure the computations are performed in the order you intend.

1- Arithmetic Operators

- Python Arithmetic Operators are;

Symbol	Operator Name	Description	Example	Result
+	Addition	Adds the values on either side of the operator and calculate a result.	4+4	8
-	Subtraction	Subtracts values of right side operand from left side operand.	4-2	2
*	Multiplication	Multiplies the values on both sides of the operator.	4*2	8
/	Division	Divides left side operand with right side operand.	4/2	2
%	Modulus	It returns the remainder by dividing the left side operand with right side operand	10%3	1
**	Exponent	Calculates the exponential power	2**3	8
//	Floor Division	Here the result is the quotient in which the digits after decimal points are not taken into account.	10//3	3

2- Assignment Operators

- Python Assignment Operators are;

Symbol	Operator Name	Description	Example
=	Equal	Assigns the values of right side operand to left side operand.	a=3 a=a+1 print(a)
+=	Add AND	Adds right side operand value to the left side operand value and assigns the results to the left operand.	b=4 b+=2 print(b)
-=	Subtract AND	Subtracts right side operand value to the left side operand value and assigns the results to the left operand.	c=6 c-=2 print(c)

2- Assignment Operators

Symbol	Operator Name	Description	Example
*=	Multiply AND	Similarly does their respective operations and assigns the operator value to the left operand.	d=10 d*=2 print(d)
/=	Division AND		e=10 e/=2 print(e)
%=	Modulus AND		f=10 f%=3 print(f)
**=	Exponent AND		g=4 g**=2 print(g)
//=	Floor Division AND		h=10 h//=3 print(h)

Boolean expressions

- A *boolean expression* is an expression that is either True or False.

- The following examples use the operator ==, which compares two operands and produces True if they are equal and False otherwise.

Statement	Output
5==5	True
5==6	False

- True and False are special values that belong to the class bool; they are not strings.

Statement	Output
type(True)	bool
type(False)	bool

Boolean expressions

- Python symbols are different from the mathematical symbols for the same operations.
- Common error is to use a single equal sign (=) instead of a double equal sign (==).
- Remember that = is an assignment operator and == is a comparison operator.
- There is no such thing as =< or =>.

3- Comparison (Relational) Operators

- Python Comparison Operators are;

Symbol	Operator Name	Description	Example
==	Double Equal	If the two value of its operands are equal, then the condition becomes true, otherwise false	a=2 b=2 a==b
!=	Not Equal To	If two operands values are not equal, then condition becomes true.	c=3 d=4 c!=d
>	Greater Than	If the value of the left-hand operand is greater than the value of right-hand operand, the condition becomes true.	e=6 f=4 e>f
<	Less Than	If the value of the left-hand operand is less than the value of right operand, then condition becomes true.	g=7 h=4 g<h
<=	Less Than Equal To	If the value of the left-hand operand is less than or equal to the value of right-hand operand, the condition becomes true.	i=9 j=9 i<=j
>=	Greater Than Equal To	If the value of the left-hand operand is greater than or equal to the value of right-hand operand, the condition becomes true.	k=10 l=10 k>=l

4- Logical Operators

- Python Logical Operators are;

NOT Logical Operator	
x	not x
FALSE	TRUE
TRUE	FALSE

AND Logical Operator		
X	Y	X and Y
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

OR Logical Operator		
X	Y	X and Y
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

- **NOT Logical Operator:**
- If Input is FALSE then output is TRUE.
- If Input is TRUE then output is FALSE.
- **AND Logical Operator:**
- If Both inputs are FALSE then output is FALSE.
- If Both input is TRUE then output is TRUE.
- If any input is FALSE then output is FALSE.
- **OR Logical Operator:**
- If Both inputs are FALSE then output is FALSE.
- If Bout input is TRUE then output is TRUE.
- If any input is TRUE then output is TRUE.

4- Logical Operators

Symbol	Operator Name	Description	Example
or	Logical OR	If any of the two operands are non-zero, then the condition is true.	condition1=100>10 condition1 condition2=1>2 condition2 condition1 or condition2
and	Logical AND	If both the operands are true then the condition is true.	condition1=100>10 condition1 condition2=1>2 condition2 condition1 and condition2
not	Logical NOT	It is used to reverse the logical state of its operand.	a=2 not a==1

5- Identity Operators

- For comparing memory locations of two objects, identity operators are used.
- There are two types of Python identity operators;

Symbol	Operator Name	Description	Example
is	is	The result becomes true if values on either side of the operator point to the same object and False otherwise.	umer=1 type(umer) is str
is not	is not	The result becomes False if the variables on either side of the operator points to the same object	ali="Hello" type(ali) is not str

6- Bitwise Operators

Microsoft Word Document

- These operators are used to manipulate with bits, & performs bit-by-bit operations.
- There are six types of bitwise operators supported by Python.

Symbol	Operator Name	Description	Statement	Output
&	Binary AND	This operator copies the bit to the result if it exists in both operands.	a=16 b=14 a&b	0
|	Binary OR	This operator copies the bit if it exists in either of the operands.	a=16 b=14 a|b	30
^	Binary XOR	This operator copies the bit if it is set in one operand but not both.	a=16 b=14 a^b	30

6- Bitwise Operators

Symbol	Operator Name	Description	Statement	Output
~	Binary 1s Complement	This is a unary operator and has the ability of 'flipping' bits.	a=16 b=14 ~a	-17
<<	Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand using this operator.	a=16 b=14 b<<2	56
>>	Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand using this operator.	a=16 b=14 b>>2	3

7- Membership Operators

- Python Membership Operators are;

Symbol	Operator Name	Description	Statement	Output
in	in	The result of this operation becomes True if it finds a value in a specified sequence & False otherwise.	a=[1,2,3] b=[3] b in a	FALSE
not in	not in	result of this operation becomes True if it doesn't find a value in a specified sequence & False otherwise.	a=[1,2,3] b=[3] b not in a	TRUE

String Operations

Rules Applied in String Construction

- 1- The quotes at the beginning and end of a string should be both double quotes or both single quote. They can not be mixed.
- 2- Double quotes can be inserted into a string starting and ending with single quote.
- 3-Single quote can be inserted into a string starting and ending with double quotes.
- 4- Double quotes can not be inserted into a string starting and ending with double quotes.
- 5- Single quote can not be inserted into a string starting and ending with single quote.

Rules Applied in String Construction

Statement	Validity	Rule
a ="Start and end with single quote"	valid	1
b ='Start and end with single quote'	valid	1
c ="Start and end with single quote'	invalid	1
d='Double quotes can be inserted" into a string starting and ending with single quote'	valid	2
e="Single quote can be inserted' into a string starting and ending with double quotes"	valid	3
f="Double quotes can not be inserted" into a string starting and ending with double quotes"	invalid	4
g='Single quote can not be inserted' into a string starting and ending with single quote'	invalid	5

Rules Applied in String Construction

Statement	Validity	Rule
h = "'!$*#@ you!' she replied"	valid	
i = "'!$*#@ you!" she replied'	valid	
j = "Once again he asked \"How long is a piece of string?\""	valid	
k = 'Once again he asked \'How long is a piece of string?\'	valid	
l="""" Hello, how are you?, whats going on? """"	valid	

Escape Sequences

Escape Sequence	Description	Example
\newline	Backslash and newline ignored	l="""" Hello,\ how are you?, whats going on? """"
\a	ASCII Bell (BEL)	a="Umer Saeed\a"
\b	ASCII Backspace (BS)	a="F2017313014\b @umt.edu.pk"
\n	ASCII Linefeed (LF)	l="""" Hello,\n how are you?, whts going on? """"
\t	ASCII Horizontal Tab (TAB)	a="hotmail\t hotmail yahoomail"

String Example

Statements	Comments
str = 'Hello Python'	
print (str)	# this will print the complete string
print (str[0])	# this will print the first character of the string
print (str[2:8])	# this will print the characters starting from 3rd to 8th
print (str[3:])	# this will print the string starting from the 4th character
print (str * 3)	# this will print the string three times
print (str + "String")	# this will print the concatenated string
print (str [:8]+ "Python")	# Updating a String Value
len(str)	# Length of the string
first='100' second='150' print(first+second)	100150

- The + operator works with strings, but it is not addition in the mathematical sense.
- Instead it performs *concatenation*, which means joining the strings by linking them end to end.

String Formatting Operator

- The % symbol has special meaning in Python strings.
- It can be used as a placeholder for another value to be inserted into the string
- The % symbol is a prefix to another character which defines the type of value to be inserted.

Statements	Output
print("Hello %s, you scored %i out of %i" % ("Homer", 90, 100))	Hello Homer, you scored 90 out of 100

- So we used %s where we wanted to insert a string and %i for an integer.
- The values are provided after the % after the end of the string.

String Formatting Operator

Character	Description
%c	Character.
%s	String conversion via str() prior to formatting.
%i	Signed decimal integer.
%d	Signed decimal integer.
%u	Unsigned decimal integer.
%o	Octal integer.
%x	Hexadecimal integer using lowercase letters.
%X	Hexadecimal integer using uppercase letters.
%e	Exponential notation with lowercase e.
%E	Exponential notation with uppercase e.
%f	Floating point real number.
%g	The shorter of %f and %e.
%G	The shorter of %f and %E.

Asking the user for input

Asking the user for input

- Sometimes we would like to take the value for a variable from the user via their keyboard.
- Python provides a built-in function called input that gets input from the keyboard.
- When this function is called, the program stops and waits for the user to type something.
- When the user presses Return or Enter, the program resumes and input returns what the user typed as a string.
- **Example:**

Statements	Input	Output
inp = input()	Umer Saeed	
print(inp)		Umer Saeed

Asking the user for input

- Before getting input from the user, it is a good idea to print a prompt telling the user what to input.

- You can pass a string to input to be displayed to the user before pausing for input.

Statements	Input	Output
name = input('What is your name?')	Umer Saeed	
print(name)		Umer Saeed

Statements	Input	Output
name = input('What is your name?\n')	Umer Saeed	
print(name)		Umer Saeed

- The sequence \n at the end of the prompt represents a *newline*, which is a special character that causes a line break.

- why the user's input appears below the prompt.

Asking the user for input

- If you expect the user to type an integer, you can try to convert the return value to int using the int() function:

Statement	prompt = 'What...is the airspeed velocity of an unladen swallow?\n'
Input	speed = input(prompt)
Statement	int(speed)
Output	int(speed)+5

- But if the user types something other than a string of digits, you get an error.

- ValueError: invalid literal for int() with base 10: 'Corvit'

Comments in Python Language

Comments in Python Language

- As programs get bigger and more complicated, they get more difficult to read.
- Formal languages are dense, and it is often difficult to look at a piece of code and figure out what it is doing, or why.
- For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing.
- These notes are called *comments*, and in Python they start with the # symbol
- # My first program in Python Programming
- In this case, the comment appears on a line by itself. You can also put comments at the end of a line
- percentage = (minute * 100) / 60 # percentage of an hour

Comments in Python Language

- Everything from the # to the end of the line is ignored; it has no effect on the program.
- It is reasonable to assume that the reader can figure out *what* the code does; it is much more useful to explain *why*.
- This comment is redundant with the code and useless:

```
v = 5        # assign 5 to v
```

- This comment contains useful information that is not in the code:

```
v = 5        # velocity in meters/second
```

- Good variable names can reduce the need of the comments.
- but long names can make complex expressions hard to read, so there is a trade-off.

Conditional Execution

Conditional Execution

- Decisions in a program are used when the program has conditional choices to execute code block.
- Let's take an example of traffic lights, where different colors of lights lit up at different situations based on the conditions of the road or any specific rule.
- It is the prediction of conditions that occur while executing a program to specify actions.
- Multiple expressions get evaluated with an outcome of either TRUE or FALSE.
- These are logical decisions and Python also provides decision-making statements that to make decisions within a program for an application based on the user requirement.

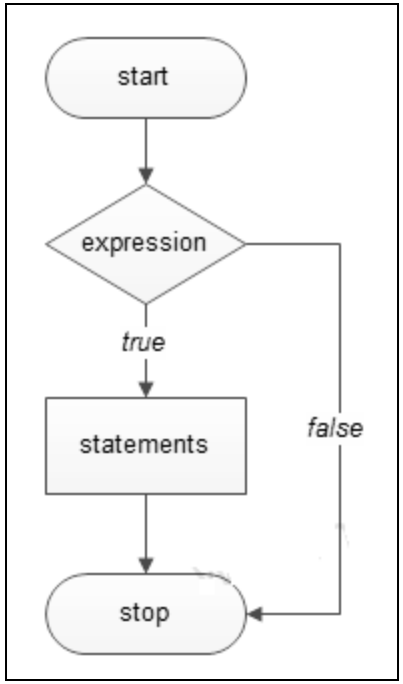
Conditional Execution

- Python Conditional Statements are;
  1. if Statements
  2. if else Statements
  3. Nested Statements

if Statement

Statement	Description
if Statements	It consists of a Boolean expression which results is either TRUE or FALSE followed by one or more statements.

- The decision-making structures can be recognized and understood using flowcharts.



if Statement

- The boolean expression after the if statement is called the *condition*.
- We end the if statement with a colon character (:)
- and the line(s) after the if statement are indented.
- **Syntax:**

```
if expression:
   #execute your code
```

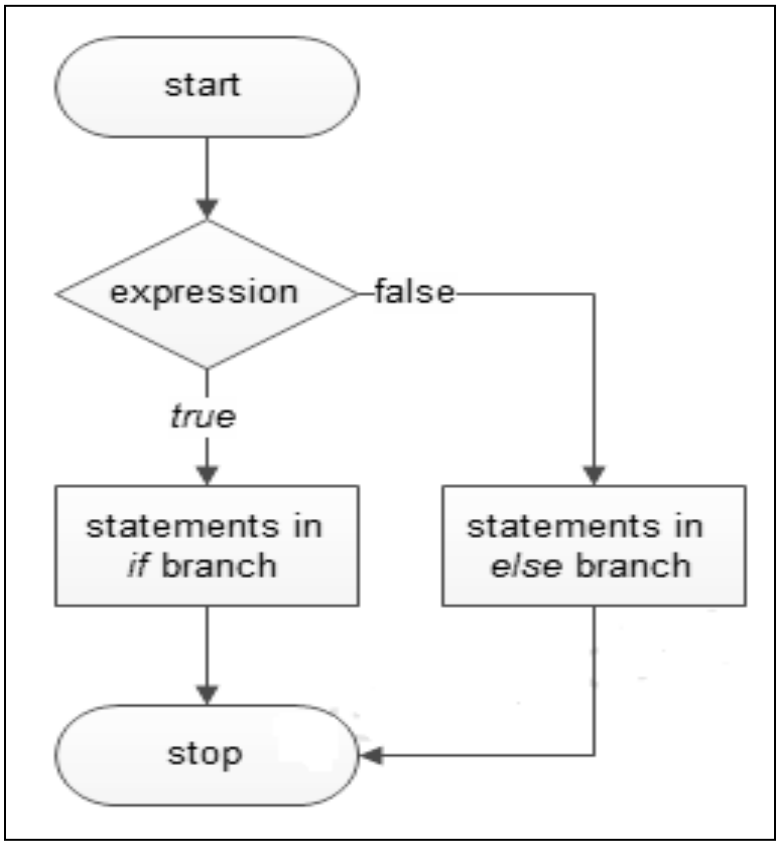
- If the logical condition is true, then the indented statement gets executed.
- If the logical condition is false, the indented statement is skipped.

if Statement

Example 1	Example 2
a = 15 if a > 10: print("Pass")	a = 9 if a > 10: print("Pass")
Output1	Output2
Pass	Skipped
Example 3	**Example 4**
x=-5 if x < 0 : print("Fail")	a = 6 if a > 10 or a<20: print("A+")
Output3	Output4
Fail	A+

Alternative execution

- The if else structures can be recognized and understood using flowcharts.



Alternative execution

- A second form of the if statement is *alternative execution*, in which there are two possibilities and the condition determines which one gets executed.
- Since the condition must either be true or false, exactly one of the alternatives will be executed.
- The alternatives are called *branches*, because they are branches in the flow of execution.
- **Syntax:**

```
if expression:
   #execute your code
else:
   #execute your code
```

Alternative execution

Example 1	Example 2
a = 15; b = 20; if a > b: print("a is greater") else: print("b is greater")	x=25 if x%2 == 0 : print('x is even') else : print('x is odd')
Output1	**Output2**
b is greater	x is odd

- If the remainder when x is divided by 2 is 0, then we know that x is even, and the program displays a message to that effect. If the condition is false, the second set of statements is executed.

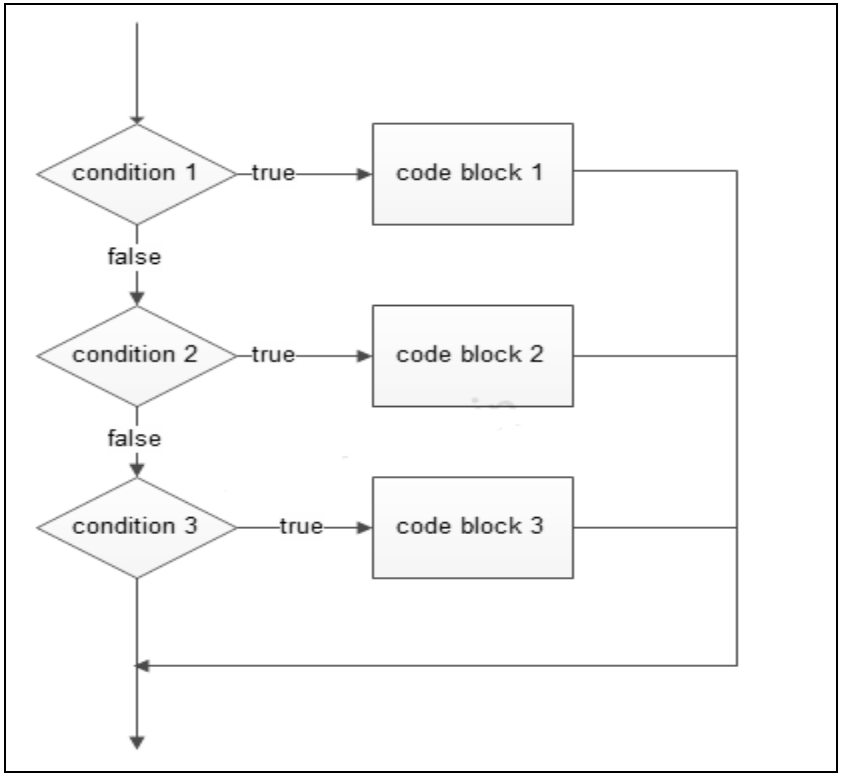
Chained conditionals (elif Statements)

Statement	Description
if else Statements	It also contains a Boolean expression. The if statement is followed by an optional else statement & if the expression results in FALSE, then else statement gets executed. It is also called alternative execution in which there are two possibilities of the condition determined in which any one of them will get executed.

- Sometimes there are more than two possibilities and we need more than two branches.
- One way to express a computation like that is a *chained conditional*.
- elif is an abbreviation of “else if.”
- Again, exactly one branch will be executed.

Chained conditionals (elif Statements)

- There is no limit on the number of elif statements.
- If there is an else clause, it has to be at the end, but there doesn't have to be one.



Chained conditionals (elif Statements)

- **Syntax:**

```
if expression:
   #execute your code
elif expression:
   #execute your code
else:
   #execute your code
```

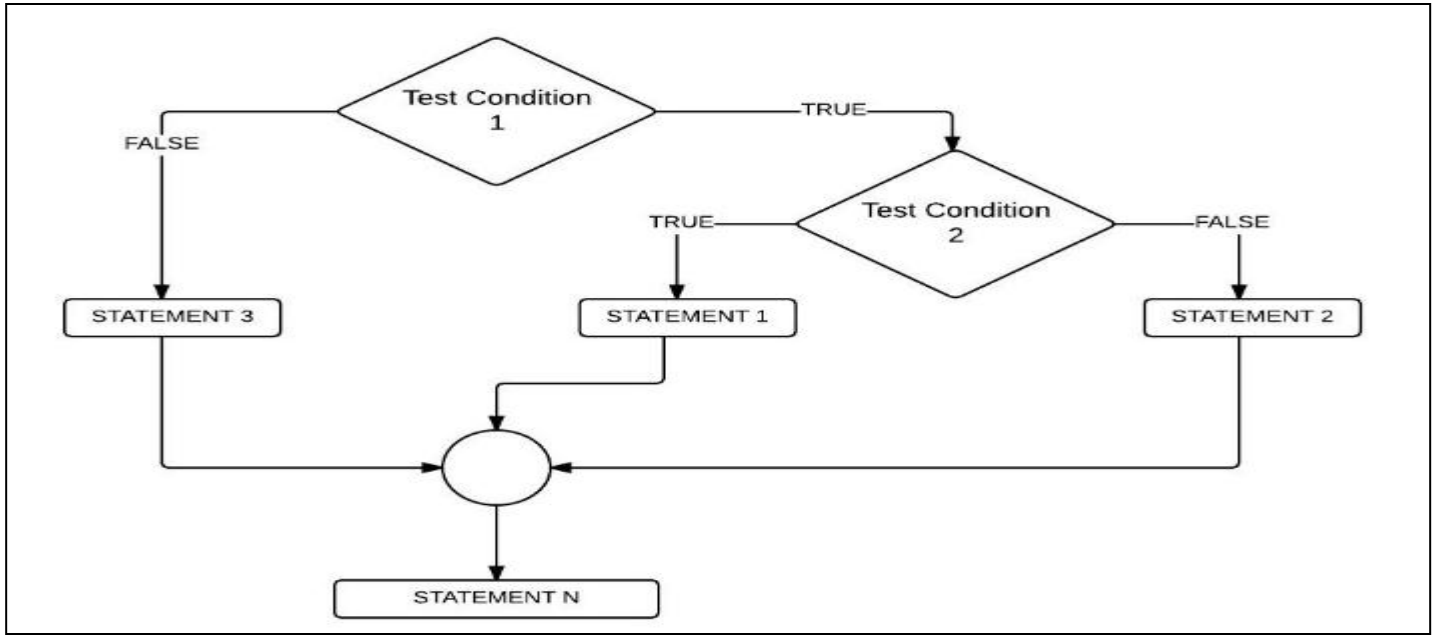
Example 1	Example 2
a = 15;b = 15; if a > b: print("a is greater") elif a == b: print("both are equal") else: print("b is greater")	choice ='a' if choice == 'a': print('Bad guess') elif choice == 'b': print('Good guess') elif choice == 'c': print('Close, but not correct')
Output1	Output2
both are equal	Bad guess

Chained conditionals (elif Statements)

- Each condition is checked in order. If the first is false, the next is checked, and so on.
- If one of them is true, the corresponding branch executes, and the statement ends.
- Even if more than one condition is true, only the first true branch executes.

Nested Statements

Statement	Description
Nested Statements	We can implement if statement and or if-else statement inside another if or if - else statement. Here more than one if conditions are applied & there can be more than one if within elif.



Nested Statements

- If the Test Condition1 is FALSE then STATEMENT3 will be executed. If Test Condition1 is TRUE then it will check for the Test Condition2, if it is TRUE then STATEMENT1 will be executed or else STATEMENT2.

- **Syntax:**

```
if (condition1):
   # Executes when condition1 is true
   if (condition2):
      # Executes when condition2 is true
   # if Block is end here
# if Block is end here
```

Nested Statements

Example
age = int(input(" Please Enter Your Age Here: "))
if age < 18:
print(" You are Minor ")
print(" You are not Eligible to Work ")
else:
if age >= 18 and age <= 60:
print(" You are Eligible to Work ")
print(" Please fill in your details and apply")
else:
print(" You are too old to work as per the Government rules")
print(" Please Collect your pension!")

Python Function

Function

- Functions are reusable pieces of programs.
- They allow you to give a name to a block of statements, allowing you to run that block using the specified name anywhere in your program and any number of times. This is known as *calling* the function.
- **Syntax:**
- Functions are defined using the def keyword.
- After this keyword comes an ***identifier*** name for the function
- Followed by a pair of parentheses which may enclose some names of variables.
- and by the final colon that ends the line.
- Next follows the block of statements that are part of this function.

Function (Example)

Function	Function Re-Call	Output

def say_hello():
  # block belonging to the function
  print('Hello Class')
  print('I am Umer Saeed')
# End of function 	
	say_hello()	Hello Class
		I am Umer Saeed

	say_hello()	Hello Class I am Umer Saeed

- We define a function called say_hello using the syntax as explained above.
- This function takes no parameters and hence there are no variables declared in the parentheses.
- Parameters to functions are just input to the function so that we can pass in different values to it and get back corresponding results.
- Notice that we can call the same function twice which means we do not have to write the same code again.

Function Parameters

- A function can take parameters, which are values you supply to the function so that the function can *do* something utilizing those values.
- These parameters are just like variables except that the values of these variables are defined when we call the function and are already assigned values when the function runs.
- Parameters are specified within the pair of parentheses in the function definition, separated by commas.
- When we call the function, we supply the values in the same way.
- Note the terminology used - the names given in the function definition are called *parameters*
- whereas the values you supply in the function call are called *arguments*.

Function Parameters (Example)

Function	Function Re-Call	Output

def umer_max(a, b):
  if a > b:
  print(a, 'is maximum')
  elif a == b:
  print(a, 'is equal to', b)
  else:
  print(b, 'is maximum')
	a=4; b=6; umer_max(a, b)	6 is maximum

	x=10; y=16; umer_max(x, y)	16 is maximum
	umer_max(a=8, b=4)	8 is maximum

Function Parameters (Example)

- we define a function called print_max that uses two parameters called a and b.
- We find out the greater number using a simple if..else statement and then print the bigger number.
- The first time we call the function print_max, we directly supply the numbers as arguments
- In the second case, we call the function with variables as arguments. print_max(x, y) causes the value of argument x to be assigned to parameter a and the value of argument y to be assigned to parameter b.

Local Variables

- When you declare variables inside a function definition, they are not related in any way to other variables with the same names used outside the function - i.e. variable names are *local* to the function.

- This is called the *scope* of the variable.

- All variables have the scope of the block they are declared in starting from the point of definition of the name.

Local Variables (Example)

Function	Function Re-Call	Output

x = 50
 	
def func(x):
  print('x is', x)
  x = 2
  print('Changed local x to', x)
	func(x)	x is 50
		Changed local x to 2

x = 50
def func(x):
  print('x is', x)
  x = 2
  print('Changed local x to', x)
 	
func(x)
print('x is still', x)
	func(x)	x is 50
		Changed local x to 2
		x is still 50

Local Variables (Example)

- The first time that we print the *value* of the name *x* with the first line in the function's body, Python uses the value of the parameter declared in the main block, above the function definition.

- Next, we assign the value 2 to x. The name x is local to our function. So, when we change the value of x in the function, the x defined in the main block remains unaffected.

- With the last print statement, we display the value of x as defined in the main block, thereby confirming that it is actually unaffected by the local assignment within the previously called function.

The global statement

- If you want to assign a value to a name defined at the top level of the program (i.e. not inside any kind of scope such as functions or classes), then you have to tell Python that the name is not local, but it is *global*.
- We do this using the global statement.
- It is impossible to assign a value to a variable defined outside a function without the global statement.
- You can use the values of such variables defined outside the function (assuming there is no variable with the same name within the function).
- However, this is not encouraged and should be avoided since it becomes unclear to the reader of the program as to where that variable's definition is.
- Using the global statement makes it amply clear that the variable is defined in an outermost block.

The global statement (Example)

Function	Output

x = 50
def func():
  global x
  print('x is', x)
  x = 2
  print('Changed global x to', x)
func()
print('Value of x is', x)
y=x+1
print(y)
	x is 50
	Changed global x to 2
	Value of x is 2
	3

- The global statement is used to declare that x is a global variable - hence, when we assign a value to x inside the function, that change is reflected when we use the value of x in the main block.

- You can specify more than one global variable using the same global statement e.g. global x, y, z.

Default Argument Values

- For some functions, you may want to make some parameters *optional* and use default values in case the user does not want to provide values for them.
- This is done with the help of default argument values.
- You can specify default argument values for parameters by appending to the parameter name in the function definition the assignment operator (=) followed by the default value.
- Note that the default argument value should be a constant.

Default Argument Values (example)

- The function named say is used to print a string as many times as specified. If we don't supply a value, then by default, the string is printed just once. We achieve this by specifying a default argument value of 1 to the parameter times.

- In the first usage of say, we supply only the string and it prints the string once. In the second usage of say, we supply both the string and an argument 5 stating that we want to *say* the string message 5 times.

Function	Function Re-Call	Output1	Output2
def say(message, times=1): print(message * times)	say('Hello') say('World', 5)	Hello	WorldWorldWorldWorldWorld

- ***CAUTION:*** Only those parameters which are at the end of the parameter list can be given default argument values i.e. you cannot have a parameter with a default argument value preceding a parameter without a default argument value in the function's parameter list.
- This is because the values are assigned to the parameters by position. For example, def func(a, b=5) is valid, but def func(a=5, b) is *not valid*.

Keyword Arguments

- If you have some functions with many parameters and you want to specify only some of them, then you can give values for such parameters by naming them - this is called *keyword arguments* - we use the name (keyword) instead of the position (which we have been using all along) to specify the arguments to the function.

- There are two advantages ;
- 1- Using the function is easier since we do not need to worry about the order of the arguments.
- 2- we can give values to only those parameters to which we want to, provided that the other parameters have default argument values.

Keyword Arguments (Example)

Function	Function Re-Call	Output1	Output2	Output3
def func(a, b=5, c=10): print('a is', a, 'and b is', b, 'and c is', c)	func(3, 7) func(25, c=24) func(c=50, a=100)	a is 3 and b is 7 and c is 10	a is 25 and b is 5 and c is 24	a is 100 and b is 5 and c is 50

- The function named func has one parameter without a default argument value, followed by two parameters with default argument values.
- In the first usage, func(3, 7), the parameter a gets the value 3, the parameter b gets the value 7and c gets the default value of 10.
- In the second usage func(25, c=24), the variable a gets the value of 25 due to the position of the argument. Then, the parameter c gets the value of 24 due to naming i.e. keyword arguments. The variable b gets the default value of 5.
- In the third usage func(c=50, a=100), we use keyword arguments for all specified values. Notice that we are specifying the value for parameter c before that for a even though a is defined before c in the function definition.

The return statement

- The return statement is used to *return* from a function i.e. break out of the function.
- We can optionally *return a value* from the function as well.
- **Example-1:**

Function	Function Re-Call	Output1

def maximum(x, y):
  if x > y:
  return x
  elif x == y:
  return 'The numbers are equal'
  else:
  return y
	maximum(7, 4)	7

The return statement

Function	Function Re-Call

def basicArithmetic(x, y):
  # Do the calulations and put each result into a variable
  sum1 = x + y
  difference = x - y
  product = x * y
  quotient = x / y
  # Return each variable
  return sum1, difference, product, quotient
	basicArithmetic(6, 3)
	basicArithmetic(6, 3)[2]

DocStrings

- Python has a nifty feature called *documentation strings*, usually referred to by its shorter name *docstrings.*
- DocStrings are an important tool that you should make use of since it helps to document the program better and makes it easier to understand.
- Amazingly, we can even get the docstring back from, say a function, when the program is actually running!

DocStrings (Example)

Function	Function Re-Call	Output

def print_max(x, y):
'''Prints the maximum of two numbers.
 	
The two values must be integers.'''
# convert to integers, if possible 	
x = int(x)
y = int(y)
 	
if x > y:
print(x, 'is maximum')
else:
print(y, 'is maximum')
	print_max(3, 5)
	print(print_max.__doc__)
		5 is maximum
		Prints the maximum of two numbers.
		
		The two values must be integers.

Iteration

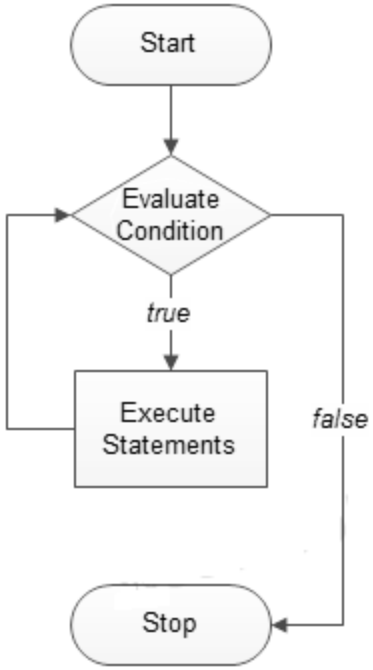
The while Statement

Loop	Description
while Loop	The loop gets repeated until the certain Boolean condition is met.

- Computers are often used to automate repetitive tasks.
- The while statement allows you to repeatedly execute a block of statements as long as a condition is true.
- Repeating identical or similar tasks without making errors is something that computers do well and people do poorly.

The while Statement

- The graphical representation of the logic behind while looping is shown below:



Syntax:

```
while expression:
    #execute your code
```

You can almost read the while statement as if it were English.

The while Statement

- **Example-1:**

- It means, “While n is greater than 0, display the value of n and then reduce the value of n by 1.

- When you get to 0, exit the while statement and display the word Blastoff!”

- More formally, here is the flow of execution for a while statement:

- 1- Evaluate the condition, yielding True or False.

- 2- If the condition is false, exit the while statement and continue execution at the next statement.

- 3- If the condition is true, execute the body and then go back to step 1.

Program	Output

n = 5
while n > 0:
  print(n)
  n = n - 1
  print('Blastoff!')
	5
	Blastoff!
	4
	Blastoff!
	3
	Blastoff!
	2
	Blastoff!
	1
	Blastoff!

The while Statement

Example-2

Program	Output

count =1
while count < 6 :
  print (count)
  count+=1
	1
	2
	3
	4
	5

- Set a counter to 1
- Enter a while loop that keeps repeating while the number is less than 6
- Each time it repeats, print the counter value to the screen
- Increment the counter by 1

A while Loop with else

- Python lets you add an else part to your loops. This is similar to using else with an if statement.

- It lets you specify what to do when/if the loop condition becomes false.

Program	Output

counter = 1
while (counter <10):
  print (counter)
  counter = counter +1
else:
  print("The loop has successfully completed!")
	1
	2
	3
	4
	5
	6
	7
	8
	9
	The loop has successfully completed!

Positioning the Counter Increment

- A common mistake that new programmers make is putting the counter increment in the wrong place. Of course, you can put it anywhere within your loop, however, bear in mind that its position can affect the output.

- Here's what happens if we modify the previous example, so that the counter increments *before* we print its value:

Program	Output

counter = 1
while (counter < 10):
  counter = counter +1 #We moved this line
  print (counter)
	2
	3
	4
	5

The while Statement

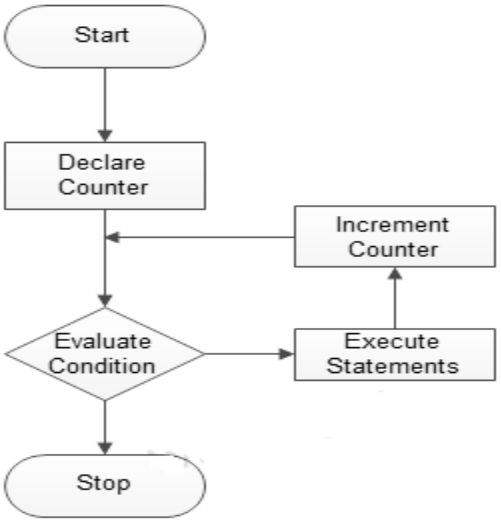
Program

```
number = 23
running = True

while running:
    guess = int(input('Enter an integer : '))

    if guess == number:
        print('Congratulations, you guessed it.')
        # this causes the while loop to stop
        running = False
    elif guess < number:
        print('No, it is a little higher than that.')
    else:
        print('No, it is a little lower than that.')
else:
    print('The while loop is over.')
    # Do anything else you want to do here

print('Done')
```

The for loop

Loop	Description
for Loop	This is traditionally used when programmers have a piece of code and wanted to repeat that 'n' number of times.

- The graphical representation of the logic behind for looping is shown below:



Syntax:

```
for iterating_var in sequence:
   #execute your code
```

The for loop

Program	Output
for x in range (0,3) : print ('Loop execution %d' % (x))	Loop execution 0 Loop execution 1 Loop execution 2
for letter in 'TutorialsCloud': print ('Current letter is:', letter)	Current letter is: T Current letter is: u Current letter is: t Current letter is: o Current letter is: r Current letter is: i Current letter is: a Current letter is: l Current letter is: s Current letter is: C Current letter is: l Current letter is: o Current letter is: u Current letter is: d

The for loop

Program	Output
planets = ["Earth", "Mars", "Neptune", "Venus", "Mercury", "Saturn", "Jupiter", "Uranus"] for i in planets: print(i)	Earth Mars Neptune Venus Mercury Saturn Jupiter Uranus

- Create a list of planets called planet.
- Enter a for loop that iterates through each item in the planet list. The irepresents each individual item in the list. This could be called anything. For example, we could've called it planet to better describe the items we're iterating against.
- For each iteration, print the current item to the screen.

A for Loop with else

- Python lets you add an else part to your loops.
- This is similar to using else with an if statement.
- It lets you specify what to do when the loop has finished.

Program	Output

planets = ["Earth", "Mars", "Neptune", "Venus", "Mercury", "Saturn", "Jupiter", "Uranus"]
 	
for i in planets:
  print(i)
else:
  print("That's all folks!")
	Earth
	Mars
	Neptune
	Venus
	Mercury
	Saturn
	Jupiter
	Uranus
	That's all folks!

Loop Control Statements

- These statements are used to change execution from its normal sequence.
- Python supports three types of loop control statements:

Control Statements	Description	Syntax
Break statement	It is used to exit a while loop or a for loop. It terminates the looping & transfers execution to the statement next to the loop.	break
Continue statement	It causes the looping to skip the rest part of its body & start re-testing its condition.	continue
Pass statement	It is used in Python to when a statement is required syntactically and the programmer does not want to execute any code block or command.	pass

The break Statement

- The break statement is used to *break* out of a loop statement i.e. stop the execution of a looping statement, even if the loop condition has not become False or the sequence of items has not been completely iterated over.

- An important note is that if you *break* out of a for or while loop, any corresponding loop else block is **not** executed.

Program	Output

while True:
  s = input('Enter something : ')
  if s == 'quit':
  break
  print('Length of the string is', len(s))
print('Done')
	Enter something : Umer Saeed
	Length of the string is 10

The break Statement

- A common mistake that new programmers make is putting the counter increment in the wrong place. Of course, you can put it anywhere within your loop, however, bear in mind that its position can affect the output.

- Here's what happens if we modify the previous example, so that the counter increments *before* we print its value:

Program	Output

counter = 1
while (counter < 10):
  counter = counter +1 #We moved this line
  print (counter)
  if counter == 5:
  break
	
	2
	3
	4
	5

The break Statement (for loop)

Program	Output

planets = ["Earth", "Mars", "Neptune", "Venus", "Mercury", "Saturn", "Jupiter", "Uranus"]
for i in planets:
  print(i)
  if i == "Venus":
  break
	Earth
	Mars
	Neptune
	Venus

The break Statement (for loop)

- Note that the else doesn't execute if you break your loop:

Program	Output

planets = ["Earth", "Mars", "Neptune", "Venus", "Mercury", "Saturn", "Jupiter", "Uranus"]
 	
for i in planets:
  print(i)
  if i == "Venus":
  break
else:
  print("That's all folks!")
	Earth
	Mars
	Neptune
	Venus

The continue Statement

- The continue statement is used to tell Python to skip the rest of the statements in the current loop block and to *continue* to the next iteration of the loop.

Program

```
while True:
    s = input('Enter something : ')
    if s == 'quit':
        break
    if len(s) < 3:
        print('Too small')
        continue
    print('Input is of sufficient length')
```

The pass Statement

- It is used in Python to when a statement is required syntactically and the programmer does not want to execute any code block or command.

Program	Output

for letter in 'TutorialsCloud':
  if letter == 'C':
  print ('Pass block')
  pass
  print ('Current letter is:', letter)
	Current letter is: T
	Current letter is: u
	Current letter is: t
	Current letter is: o
	Current letter is: r
	Current letter is: i
	Current letter is: a
	Current letter is: l
	Current letter is: s
	Pass block
	Current letter is: C
	Current letter is: l
	Current letter is: o
	Current letter is: u
	Current letter is: d

Nested Loops

Loop	Description
Nested Loops	Programmers can use one loop inside another; i.e. they can use for loop inside while or vice - versa or for loop inside for loop or while inside while.

Syntax:

```
for iterating_var in sequence:
   for iterating_var in sequence:
      #execute your code
         #execute your code
```

Program	Output

for g in range(1,6):
  for k in range(1, 3):
  print ("%d * %d = %d" % (g, k, g*k))
	1 * 1 = 1
	1 * 2 = 2
	2 * 1 = 2
	2 * 2 = 4
	3 * 1 = 3
	3 * 2 = 6
	4 * 1 = 4
	4 * 2 = 8
	5 * 1 = 5
	5 * 2 = 10

Lists

A list is a sequence

- Like a string, a *list* is a sequence of values.
- In a string, the values are characters; in a list, they can be any type.
- The values in list are called *elements* or sometimes *items*.
- There are several ways to create a new list; the simplest is to enclose the elements in square brackets ([and]).

A list is a sequence

Example	Program	Output
1	a=[10, 20, 30, 40] type(a)	list
2	b=['Umer Saeed', 'Ali Saeed', 'Ahmed Abdullah Saeed'] type(b)	list

- The first example is a list of four integers.
- The second is a list of three strings.

A list is a sequence

- The elements of a list don't have to be the same type.
- The following list contains a string, a float, an integer.

Example	Program	Output
1	c=['Umer Saeed',5,10.2] type(c)	list

- **Nested List:**
- A list within another list is *nested list*.

Example	Program	Output
1	d=['Ali Saeed',5.2,[1,3,7]] type(d)	list

A list is a sequence

- **Empty List:**
- A list that contains no elements is called an empty list.
- You can create one with empty brackets, [].

Example	Program	Output
1	name=['Umer Saeed', 'Ali Saeed', 'Ahmed Abdullah Saeed']	
	number=[98,99,100]	
	result=[]	
	print(name,number,result)	['Umer Saeed', 'Ali Saeed', 'Ahmed Abdullah Saeed'] [98, 99, 100] []

Lists are mutable

- The syntax for accessing the elements of a list is the same as for accessing the characters of a string: the bracket operator.
- The expression inside the brackets specifies the index.
- Remember that the indices start at 0:

Program	Output
domain=['hotmail', 'yahoomail', 'gmail'] domain[0]	hotmail

Lists are mutable

- Unlike strings, lists are mutable because you can change the order of items in a list or reassign an item in a list.

- When the bracket operator appears on the left side of an assignment, it identifies the element of the list that will be assigned.

Program	Output
number=[98,99,100]	
number[1]=100	
number	[98, 100, 100]

- The one-eth element of numbers, which used to be 99, is now 100.

- You can think of a list as a relationship between indices and elements. This relationship is called a *mapping*; each index “maps to” one of the elements.

Lists are mutable

- List indices work the same way as string indices:

Sr No	List indices work	Program	Output
1	Any integer expression can be used as an index.	university=['UET','UMT','UCP'] university[0]	UET
2	If you try to read or write an element that does not exist, you get an IndexError.	university=['UET','UMT','UCP'] university[5]	IndexError: list index out of range
3	If an index has a negative value, it counts backward from the end of the list.	university=['UET','UMT','UCP'] university[-2]	UMT

Lists are mutable

- The in operator also works on lists.

Example	Program	Output
1	university=['UET','UMT','UCP'] 'UET' in university	TRUE
2	university=['UET','UMT','UCP'] 'GCU' in university	FALSE

List operations

- The + operator concatenates lists:

Statements	Output
a = [1, 2, 3]	
b = [4, 5, 6]	
c = a + b	
c	[1, 2, 3, 4, 5, 6]

- Similarly, the * operator repeats a list a given number of times:

Example	Statements	Output
1	[0] * 4	[0, 0, 0, 0]
2	[1, 2, 3] * 3	[1, 2, 3, 1, 2, 3, 1, 2, 3]

- The first example repeats four times.
- The second example repeats the list three times.

List slices

- The slice operator also works on lists:

Example	Statements	Output
	t = ['a', 'b', 'c', 'd', 'e', 'f']	
1	t[1:3]	['b', 'c']
2	t[:4]	['a', 'b', 'c', 'd']
3	t[3:]	['d', 'e', 'f']

- Since lists are mutable, it is often useful to make a copy before performing operations that fold, spindle, or mutilate lists.

Example	Statements	Output
	t = ['a', 'b', 'c', 'd', 'e', 'f']	
1	t[:]	['a', 'b', 'c', 'd', 'e', 'f']

List slices

- A slice operator on the left side of an assignment can update multiple elements:

Statements	Output
t = ['a', 'b', 'c', 'd', 'e', 'f']	
t[1:3] = ['x', 'y']	
print(t)	['a', 'x', 'y', 'd', 'e', 'f']

- Python provides methods that operate on lists.

- **append**

Statements	Output
t = ['a', 'b', 'c']	
t.append('d')	
t	['a', 'b', 'c', 'd']

List slices

- **extend**

Statements	Output
t1 = ['d', 'a', 'e']	
t2 = ['b', 'c']	
t1.extend(t2)	
t1	['d', 'a', 'e', 'b', 'c']

- This example leaves t2 unmodified.

- **sort**
- sort arranges the elements of the list from low to high:

Statements	Output
t1 = ['d', 'a', 'e']	
t2 = ['b', 'c']	
t1.extend(t2)	
t1.sort()	
t1	['a', 'b', 'c', 'd', 'e']

List slices

- sort arranges the elements of the list from high to low:

Statements	Output
t1 = ['d', 'a', 'e']	
t2 = ['b', 'c']	
t1.extend(t2)	
t1.sort(reverse=True)	
t1	['e', 'd', 'c', 'b', 'a']

- Most list methods are void; they modify the list and return None. If you accidentally write t = t.sort(), you will be disappointed with the result.

- **How to solve this solution?**

Statements	Output
t1 = ['d', 'a', 'e']	
t2 = ['b', 'c']	
t1.extend(t2)	
t1s=sorted(t1,reverse=False)	
t1s	['a', 'b', 'c', 'd', 'e']

Deleting elements

- There are several ways to delete elements from a list.

- **pop**
- If you know the index of the element you want, you can use pop:

Statements	Output
t = ['a', 'b', 'c']	
a=t.pop(2)	
print(a)	c
print(t)	['a', 'b']

- pop modifies the list and returns the element that was removed.
- If you don't provide an index, it deletes and returns the last element.

Statements	Output
t = ['a', 'b', 'c','d','e']	
a=t.pop()	
print(a)	e
print(t)	['a', 'b', 'c', 'd']

Deleting elements

- **del operator**
- If you don't need the removed value, you can use the del operator, To remove more than one element, you can use del with a slice index. As usual, the slice selects all the elements up to, but not including, the second index.

Statements	Output
t = ['a', 'b', 'c','d','e']	
del t[1:3]	
print(t)	['a', 'd', 'e']

- **remove**
- If you know the element you want to remove (but not the index), you can use remove:

Statements	Output
t = ['a', 'b', 'c']	
t.remove('b')	
print(t)	['a', 'c']

Lists and functions

- There are a number of built-in functions that can be used on lists that allow you to quickly look through a list without writing your own loops:

Statements	Output
nums = [3, 41, 12, 9, 74, 15]	
len(nums)	6
sum(nums)	154
max(nums)	74
min(nums)	3
print(sum(nums)/len(nums))	25.66667

- The sum() function only works when the list elements are numbers.

- The other functions (max(), len(), etc.) work with lists of strings and other types that can be comparable.

Lists and strings

- A string is a sequence of characters and a list is a sequence of values, but a list of characters is not the same as a string.

- To convert from a string to a list of characters, you can use list:

Statements	Output
name = 'Umer Saeed'	
t = list(name)	
print(t)	['U', 'm', 'e', 'r', ' ', 'S', 'a', 'e', 'e', 'd']

- Because list is the name of a built-in function, you should avoid using it as a variable name.

- I also avoid the letter l because it looks too much like the number 1. So that's why I use t.

Lists and strings

- **split method**
- The list function breaks a string into individual letters.
- If you want to break a string into words, you can use the split method:

Statements	Output
message = 'Hello, My name is Umer Saeed'	
t = message.split()	
print(t)	['Hello,', 'My', 'name', 'is', 'Umer', 'Saeed']
t[2]	'name'

- Once you have used split to break the string into a list of words, you can use the index operator (square bracket) to look at a particular word in the list.

Lists and strings

- ***delimiter***
- You can call split with an optional argument called a *delimiter* that specifies which characters to use as word boundaries.

Statements	Output
message = 'Hello,My,name,is,Umer,Saeed'	
delimiter = ','	
t = message.split(delimiter)	
print(t)	['Hello', 'My', 'name', 'is', 'Umer', 'Saeed']

Lists and strings

- ***join***
- Join is the inverse of split.
- It takes a list of strings and concatenates the elements.
- join is a string method, so you have to invoke it on the delimiter and pass the list as a parameter.

Statements	Output
message = ['Hello', 'My','name', 'is', 'Umer', 'Saeed']	
delimiter = ' '	
delimiter.join(message)	Hello My name is Umer Saeed'

- In this case the delimiter is a space character, so join puts a space between words.
- To concatenate strings without spaces, you can use the empty string, “”, as a delimiter.

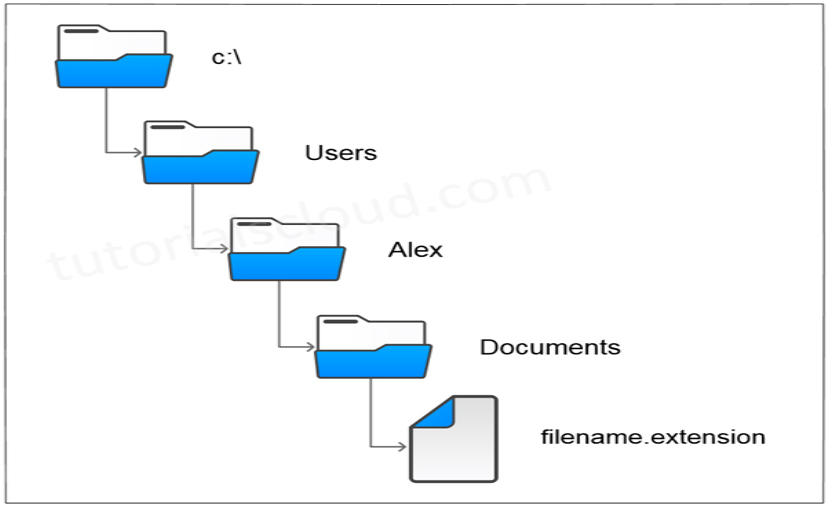
Python File Handling

Python File Handling

- All programs need the input to process and output to display data.
- And everything needs a file as name storage compartments on computers that are managed by OS.
- Though variables provide us a way to store data while the program runs, if we want out data to persist even after the termination of the program, we have to save it to a file.

File and its path

- There are always two parts of a file in the computer system;
- 1- the filename
- 2- its extension
- Also, the files have two key properties -
- 1- its name
- 2- the location or path, which specifies the location where the file exists
- The file name has two parts and they are separated by a dot (.) or period



File and its path Management

Statements	Comments
import os	Importing the Libraries for path/file management
os.getcwd()	Get working dir
os.chdir('D:\Python DS Umer Saeed')	Change working path(1st Method)
os.chdir('D:\Python DS Umer Saeed/1')	Change working path(2nd Method)
os.chdir('D:\\Python DS Umer Saeed\\1')	Change working path(3rd Method)
os.chdir('D:/Python DS Umer Saeed/1')	Change working path(4th Method)
os.chdir('D://Python DS Umer Saeed//1')	Change working path(5th Method)
os.listdir(os.curdir)	List Current dir
os.mkdir('test1')	Make a new dir in Current dir
os.rmdir('test1')	Delete dir
os.rename('Ali Saeed.txt','Umer_Saeed.txt')	Re-name of the file
os.remove('Umer_Saeed.txt')	Delete file from the current path