LECTURE 1
INTRODUCTION TO PYTHON PROGRAMMING

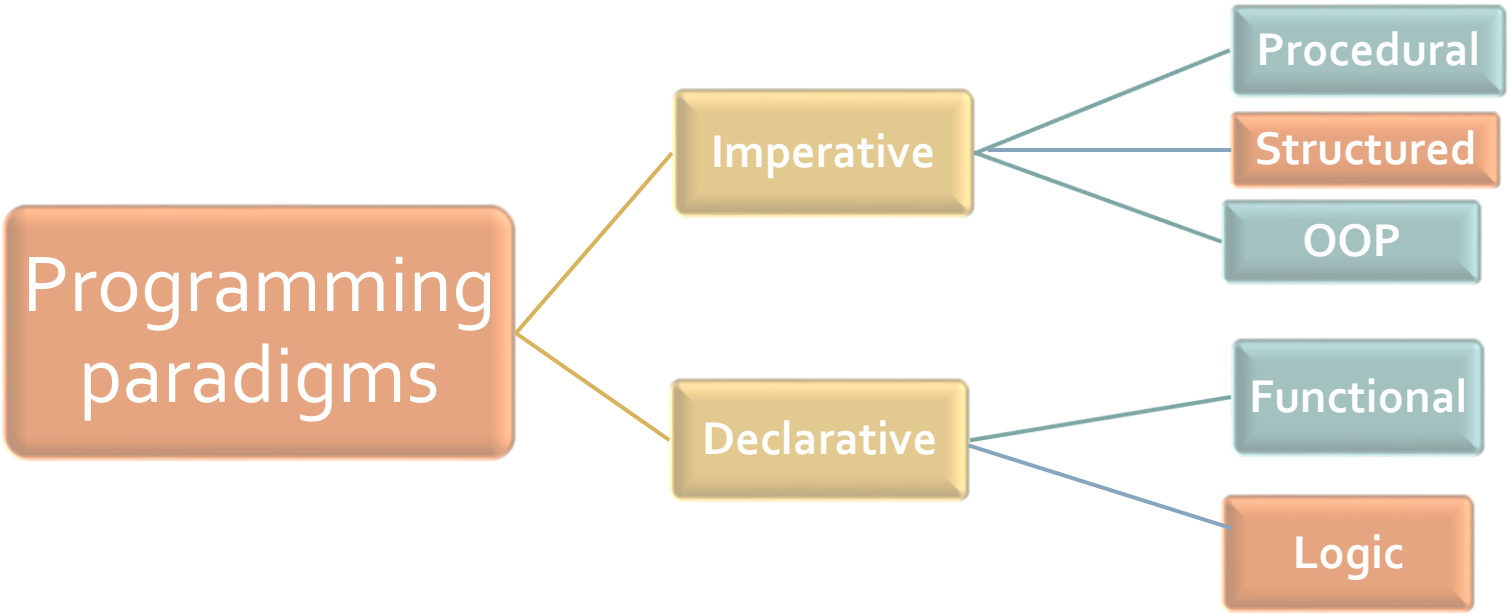
Prepared By

Dr. Wesam Mahmoud

Programming Paradigms

- Way to classify programming languages based on their features.
- Languages can be classified into multiple paradigms.

Examples of Programming Paradigms



Structured Programming

- Programming paradigm aimed at improving the clarity, quality, and development time of a computer program
- Making extensive use of subroutines, block structures, for and while loops—in contrast to using simple tests and jumps such as the goto statement
- C, C++, Java, Python are Structured Programming Languages

Why Python?

- Python is generally:
  - Comparatively easy to learn
  - Freely available
  - Cross-platform (Windows, Mac, Linux)
  - Widely used – extensive capabilities, documentation, and support
  - Access to advanced math, statistics, and database functions
  - Integrated into ArcGIS and other applications
  - Simple, interpreted language – no compilation step

Why Python?

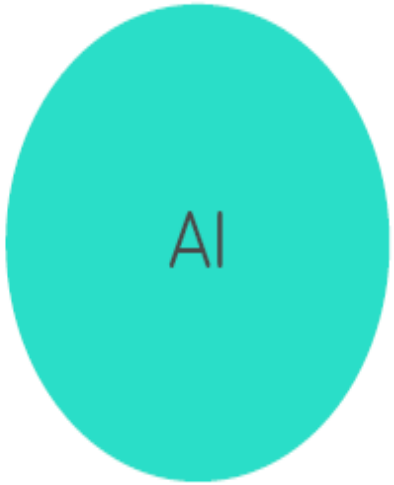
Python for everything !

Django

A python framework for web development

Pygame

A widely known python framework for game development



Almost every AI library !

Almost all of the ai technology is supported in python !

http://www.python.org

What is Python?

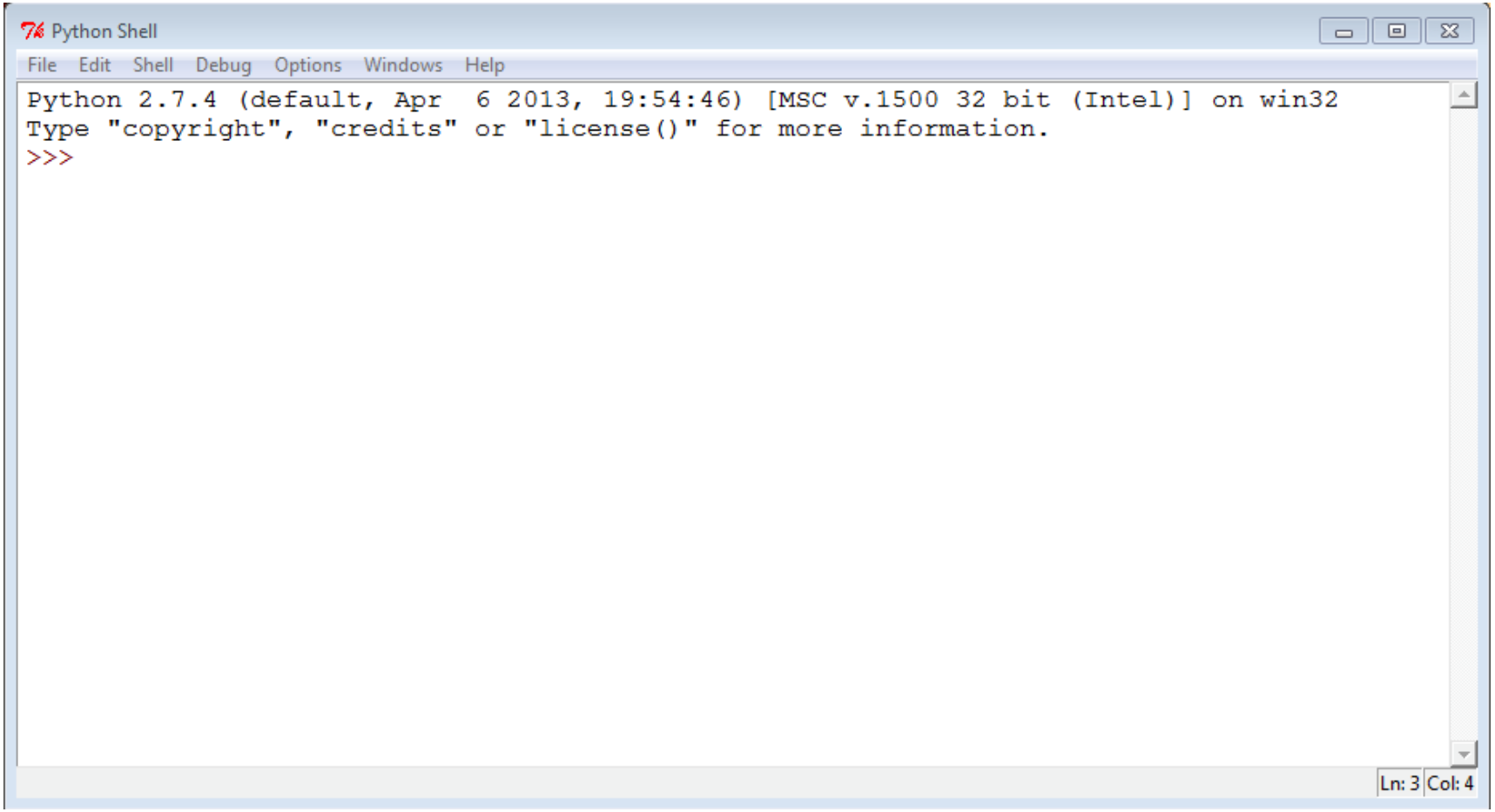
- Python is a general-purpose programming language.
- It was designed and developed to write software for a wide variety of disciplines.
- Python has been used to write applications to solve problems in
  - biology,
  - chemistry,
  - financial analysis,
  - numerical analysis,
  - robotics, and many other fields.

Python Basics

- To run Python interactively, we will use **Idle (the Python GUI)**
  - Command line interpreter – evaluates whatever you type in
  - Text editor with syntax highlighting
  - Menu commands for changing settings and running files

- Windows: Start → All Programs → Python 3.7.2 → Idle (Python GUI)

Idle on Windows



What is a variable?

- Variable is a reserved location in the memory that stores a value.
- Python is not statically typed language which means you don't need to declare the type or the variable before using them.

```
>>>planet = 'Pluto'
>>>print (planet)
Pluto
```

Variable Name

The variable name must follow the following criteria:

==========

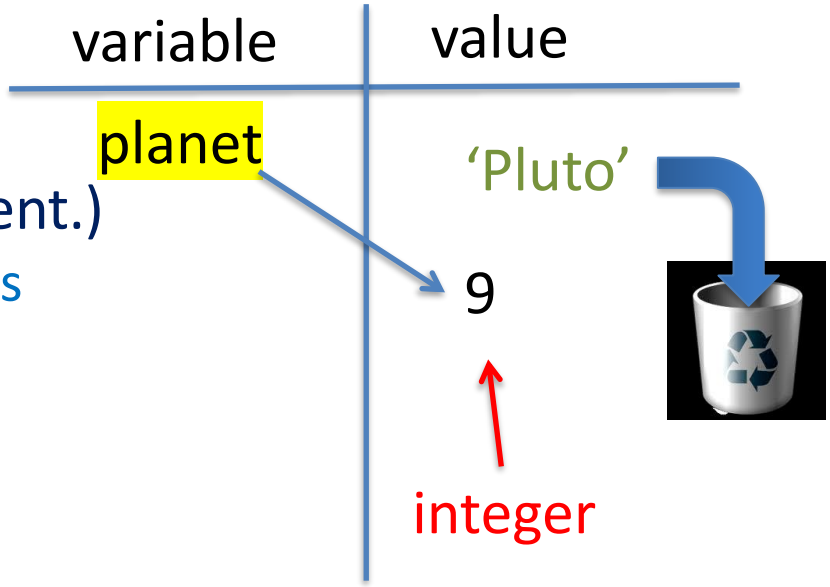
- must start with a letter or underscore
- can't start with a number
- only alpha-numeric and underscore values accepted (A-Z, a-z, 0-9, _)
- case sensitive (SCI is different from sci)
- it can be one letter (x) or a name with a meaning (my_name)

Variables

- You must assign a value to a variable before using it

>>>planet = ‘Pluto’

>>>planet = 9



Garbage Collection:

(Form of automatic memory management.)

➢ Python collects the garbage and recycles the memory (e.g., ‘Pluto’)

Multi-Line Statements

- Statements in Python typically end with a new line. Python, however, allows the use of the line continuation character (\) to denote that the line should continue.

For example :

```
Total = x + \
        y + \
        z
```

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously

For example

```
>>> a = b = c = 1
```

You can also assign multiple objects to multiple variables.

For example

```
>>> a, b, c = 1, 2, "john"
```

Values Do Have Types

- A variable is created when you first assign a value to it.
- To get the variable type after you declared it, you can use

type(variable name)

```
>>> x=7.8
>>> type(x)
```

- Check Variable Type Using isinstance()

(checks if the variable type is true or false)

```
>>> isinstance(x,str)
False
>>>string = 'two'
>>>number = 3
>>>print (string * number)          # Output???????
>>>print (string + number)          # Output???????
```

'two3' ????If so, then what is the result of '2' + '3'

- Should it be the string '23'
- Should it be the number 5
- Should it be the string '5'

```
>>>string = ‘two’
>>>number = 3
>>>print (string * number)  #Repeated concatenation
twotwotwo
>>>print  (string + number)
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    print string + number
TypeError: cannot concatenate 'str' and 'int' objects
```

Python's keywords.

- The interpreter uses keywords to recognize the structure of the program, and they cannot be used as variable names.
- Python 3 has these keywords:

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Types of Operators

Arithmetic Operators
Comparison (Relational) Operators
Bitwise operators
Assignment Operators
Logical Operators
Membership Operators
Identity Operators

Arithmetic in Python

Addition	**+**	**35 + 22**	**57**
		'Py' + 'thon'	**'Python'**
Subtraction	**-**	**35 - 22**	**13**
Multiplication	*****	**3 * 2**	**6**
		'Py' * 2	**'PyPy'**
Division	**/**	**3.0 / 2**	**1.5**
		3 / 2	**1**
Exponentiation	******	**2 ** 0.5**	**1.41421356…**

//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 = 4 and 9.0//2.0 = 4.0

x % y	Modulo operator
-x	Unary minus
+x	Unary plus

(Compound assignment operators)

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

(Comparison / Relational Operators)

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!= b) is true.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

3 < 5	True	Less than
3 != 5	True	Not equal to
3 == 5	False	Equal to (Notice double ==)
3 >= 5	False	Greater than or equal to

Chaining Comparison Operators

Consider the following comparison operators, which are connected with the AND operator:

```
x<=y and y<=z
```

These comparison operators can be chained as:

```
x<=y<=z
```

1 < 3 < 5	True	Multiple comparisons

Bitwise Operations

Every numerical that is entered in a computer is internally represented in the form of binary digits. For instance, the decimal value 25 is internally represented in the form of binary digits as 11001. The bitwise operators operate on these binary digits to give desired results.

- **Note**
- The shifting and bitwise operators can only be applied to integers and long integers.
- Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60; and b = 13; Now in binary format they will be as follows

a= 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

- Pyhton's built-in function bin() can be used to obtain binary representation of an integer number.

Bitwise Operators

Operator	Description	Example
& Binary AND	Operator copies a bit to the result, if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit, if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit, if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operand's value is moved left by the number of bits specified by the right operand.	a << = 240 (means 1111 0000)
>> Binary Right Shift	The left operand's value is moved right by the number of bits specified by the right operand.	a >> = 15 (means 0000 1111)

(Logical Operators)

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is False.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is True.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is True.

- Examples

`not (5 == 5)  # result is False`	هل ليست (5 تساوي 5) ؟
`not (5 >= 6)  # result is True`	هل ليست (5 أكبر من أو تساوي 6) ؟

`(5==5) or (3>4)  # result is True`	هل (5 تساوي 5) أو (3 أكبر من 4) ؟
`(5>6) or (4==1)  # result is False`	هل (5 أكبر من 6) أو (4 تساوي 1) ؟

`(5==5) and (3>4) # result is False`	هل (5 تساوي 5) و (3 أكبر من 4) ؟
`(5<6) and (4!=1) # result is True`	هل (5 أقل من 6) و (4 لا تساوي 1) ؟

```
5 or 6          # result is 5
0 or 2          # result is 2
2 or True       # result is 2
"Hi" and 12     # result is 12
0 and 2         # result is 0
2 and True      # result is True
```

Python Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below-

Operator	Description	Example
in	Evaluates to true, if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true, if it does not find a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Example:

```
>>> A=[1, 2, 's', 7, '9', 6.54]
                    # true
                    # false
```

Comments

- As programs get bigger and more complicated, they get more difficult to read.
- A hash sign (#) that is not inside a string literal is the beginning of a comment.
- All characters after the #, up to the end of the physical line, are part of the comment and the Python interpreter ignores them.
- # First comment
- print ("Hello, Python!")
- You can type a comment on the same line after a statement or expression
- Name = "Madisetti" # This is again comment
- Python does not have multiple-line commenting feature. You have to comment each line individually as follows-
- # This is a comment.
- # This is a comment, too.

Standard Data Types

- The data stored in memory can be of many types.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types:

❖ Numbers
❖ String
❖ List
❖ Tuple
❖ Dictionary
❖ Sets

Python Numbers

- Number data types store numeric values. Number objects are created when you assign a value to them. For example

```
>>> var1 = 1
>>> var2 = 10
```

- You can also delete the reference to a number object by using the del statement. The syntax of the del statement is −

```
>>> del var1,var2,var3...., varN
```

- You can delete a single object or multiple objects by using the del statement.
- For example
- del var1
- del var1, var2
- Python supports three different numerical types:
- int (signed integers)
- float (floating point real values)
- Complex (complex numbers) A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are real numbers and j is the imaginary unit.
- The real and imaginary components of a complex object can be accessed by using its `z.real` and `z.imag` attributes.

Frequently Used Format Codes

- You can use format codes (`%`) for substituting values in the variables at the desired place in the message:
- `print ("Length is %d and Breadth is %d" %(l,b))`
- where `%d` is a format code that indicates an integer has to be substituted at its place. That is, the values in variables l and b will replace the respective format codes.
- The list of format codes is as shown in below Table

Format Code	Usage
%s	Displays in string format.
%d	Displays in decimal format
%e	Displays in exponential format.
%f	Displays in floating-point format.
%o	Displays in octal (base 8) format.
%x	Displays in hexadecimal format.
%c	Displays ASCII code.

Use Functions to Convert Between Types

For explicit conversion, the functions that you will be frequently using are int(), float(), and str().

```
>>>print (int('2') + 3)
5
```

- **Example:**

```
from math import pi
r=5
a=pi*r*r
print ("Area of the circle is", a)
print ("Area of the circle is %.2f" %a)
```

Output:

```
Area of the circle is 78.53981633974483
Area of the circle is 78.54
```

Displaying Octal and Hexa Values

- To assign an octal value to a variable, the number should be preceded by **0o**. Similarly, if a number is preceded by **0x**, it is considered a hexa value.
- Example: The following program demonstrates conversion of a decimal value into octal and hexa and vice versa.

```
>>> a, b=0o25,0x1af
print ('Value of a in decimal is',a)
c=19
print ('19 in octal is %o and in hex is %x' %(c,c))
d=oct(c)
e=hex(c)
print ('19 in octal is', d, 'and in hexa is', e)
```

Output:

```
Value of a in decimal is 21
19 in octal is 23 and in hex is 13
19 in octal is 0o23 and in hexa is 0x13
```

The full bit patterns for ASCII code

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Python Strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows either pair of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 to the end.
- [] Slice - Gives the character from the given index
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

character	**H**	**e**	**l**	**l**	**o**		**W**	**o**	**r**	**l**	**d**	**!**
index	0	1	2	3	4	5	6	7	8	9	10	11
index from the right end	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Python Strings

- Recall the len function, which gives the length of a sequence. It works on strings.
- len(str)
- For example:

```
str = 'Hello World!'
print (str) # Prints complete string
print (str[0]) # Prints first character of the string
print (str[2:5]) # Prints characters starting from 3rd to 5th
print (str[2:]) # Prints string starting from 3rd character to end
print (str[-6:]) # Prints string starting from -6 character to end
print (str[0:11:2])
print (str * 2) # Prints string two times
print (str + "TEST") # Prints concatenated string
```

String Methods and Functions

Method/Function	Description
`str()`	Returns a string representation of the object. If the argument is a string, the returned value is the same object.
`max()`	Returns the maximum alphabetical character in the string.
`min()`	Returns the minimum alphabetical character in the string.
`len()`	Counts and returns the number of characters in the sequence (string, tuple, or list) or in the mapping (dictionary).
`sorted()`	Returns the string's characters in sorted order. After it is expanded into a list of individual characters, the string is sorted and returned.
`reversed()`	Returns the string's characters in reverse

capitalize()
Capitalizes first letter of string

Python lists

- A list contains items separated by commas and enclosed within square brackets [ ]. To some extent, lists are similar to arrays in C. One of the differences between them is that all the items belonging to a list can be of different data type.
- القائمة (list) عبارة عن تسلسل مرتب قابل للتغيير حيث يمكن إضافة قيم اليها أو ازالته أو تغييرها. ويتم تعريفها باستخدام الأقواس [ ] . مثلا

- A= [-3, -2, -1, 0, 1, 2, 3] # list contains integers
- B=[3.14, 9.23, 111.11, 312.12, 1.05] # list contains floating numbers
- sea_creatures = ['shark', 'cuttlefish', 'squid', 'mantis shrimp’]
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

Python lists

- **Example:**
- `list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]`
- `tinylist = [123, 'john']`
- `print (list)` `# Prints complete list`
- `print (list[0])` `# Prints first element of the list`
- `print (list[1:3])` `# Prints elements starting from 2nd till 3rd`
- `print (list[2:])` `# Prints elements starting from 3rd element`
- `print (tinylist * 2)` `# Prints list two times`
- `print (list + tinylist)` `# Prints concatenated lists`
- `List[2]=44444` `# changes the 3rd element in list`
- `Print(list)`
- `Print(len(list)-1)` `# You can get the last element by reducing the index by 1`
- You can index from the end of a list by using negative values, where -1 is the last element, -2 is the second to last element and so on.
- `Print(len(list)-2)`

Built-in List Functions & Methods

len(list) Gives the total length of the list.
max(list) Returns item from the list with max value.
min(list) Returns item from the list with min value.
list(seq) Converts a tuple into list.

list1=list(range(5))

Python includes the following list methods-

SN	Methods with Description
1	**list.append(obj)** Appends object obj to list
2	**list.count(obj)** Returns count of how many times obj occurs in list
3	**list.extend(seq)** Appends the contents of seq to list
4	**list.index(obj)** Returns the lowest index in list that obj appears
5	**list.insert(index, obj)** Inserts object obj into list at offset index
6	**list.pop(obj=list[-1])** Removes and returns last object or obj from list
7	**list.remove(obj)** Removes object obj from list
8	**list.reverse()** Reverses objects of list in place
9	**list.sort([func])** Sorts objects of list, use compare func if given

Examples

```
list1 = ['physics', 'chemistry', 'maths']
list2=list(range(5))   #creates list of numbers between 0-4
list1.extend(list2)
Print(list1)
```

```
list1 = ['physics', 'chemistry', 'maths']
list2=list(range(5))
list1.append(list2)
print (list1)
```

```
list1 = ['physics', 'chemistry', 'maths']
list1.insert(1, 'Biology’)
```

```
list1 = ['physics', 'Biology', 'chemistry', 'maths']
list1.pop()
print ("list now : ", list1)
list1.pop(1)
```

Examples Cont.

```
list1 = ['physics', 'Biology', 'chemistry', 'maths']
list1.remove('Biology')
print ("list now : ", list1)
```

```
list1 = ['physics', 'Biology', 'chemistry', 'maths']
list1.reverse()
print ("list now : ", list1)
```

```
list1 = ['physics', 'Biology', 'chemistry', 'maths']
list1.sort()
print ("list now : ", list1)
```

By default, Python sorts

- **strings in alphabetical order**
- **numbers in ascending numerical order**

Python Tuples

- A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parenthesis.

- الصفوف(Tuples)
- يستخدم الصف (tuple) لتجميع البيانات، وهو تسلسل ثابت من العناصر وغير قابل للتغيير. الصفوف تشبه القوائم إلى حد كبير، لكنها تستخدم الأقواس ( ) . ولأنَّها غير قابلة للتغيير، فلا يمكن تغيير أو تعديل قيمها.
- تبدو الصفوف كالتالي:

- coral = ('blue coral', 'staghorn coral', 'pillar coral')
- print(coral)
- The main difference between lists and tuples is: Lists are enclosed in brackets [ ] and their elements and size can be changed, while tuples are enclosed in parentheses ( ) and cannot be updated. Tuples can be thought of as **read-only** lists.
- To write a tuple containing a single value you have to include a comma, even though there is only one value.
- `tup1 = (50,)`

Python Tuples

- `tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )`
- `tinytuple = (123, 'john')`
- `print (tuple) # Prints complete tuple`
- `print (tuple[0]) # Prints first element of the tuple`
- `print (tuple[1:3]) # Prints elements starting from 2nd till 3rd`
- `print (tuple[2:]) # Prints elements starting from 3rd element`
- `print (tuple[:2]) # Prints elements starting from 1st element to 2nd element`
- `print (tuple[:])` # **To make a *copy* of an entire sequence**
- `print (tinytuple * 2) # Prints tuple two times`
- `print (tuple + tinytuple) # Prints concatenated tuple`
- `tuple[2] = 1000 # Invalid syntax with tuple`
- `list[2] = 1000 # Valid syntax with list`

Basic Tuples Operations

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1,2,3) : print (x, end=' ')	1 2 3	Iteration

Built-in Tuple Functions

Function with Description
len(tuple) Gives the total length of the tuple.
max(tuple) Returns item from the tuple with max value.
min(tuple) Returns item from the tuple with min value.
tuple(seq) Converts a list into tuple.

sorted() returns a copy of a tuple in order from smallest to largest, leaving the tuple unchanged

T1=(7,-3,84,93,22)

Sorted(T1)

Examples

- Example1
- tuple1, tuple2 = (123, 'xyz', 'zara'), (456, 'abc')
- print ("First tuple length : ", len(tuple1))
- print ("Second tuple length : ", len(tuple2))
- Example2
- tuple1, tuple2 = ('maths', 'che', 'phy', 'bio'), (456, 700, 200)
- print ("Max value element : ", max(tuple1))
- print ("Max value element : ", max(tuple2))
- print ("min value element : ", min(tuple1))
- Example3
- list1= ['maths', 'che', 'phy', 'bio']
- tuple1=tuple(list1)
- print ("tuple elements : ", tuple1)

- **Think:** How to reverse a tuple????????

Python Dictionary

- Python's dictionaries are kind of hash-table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces { } and values can be assigned and accessed using square braces [].
- Dict ={'name': 'Sammy', 'animal': 'shark', 'color': 'blue', 'location': 'ocean'}
- **Updating Dictionary**
- You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry
- Example:
- `dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}`
- `dict['Age'] = 8;  # update existing entry`
- `dict['School'] = "DPS School“  # Add new entry`

Python Dictionary

- **Delete Dictionary Elements**
- You can either remove individual dictionary elements or clear the entire contents of a dictionary.
- Example:

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name']                    # remove entry with key 'Name'
dict.clear()                        # remove all entries in dict
del dict                            # delete entire dictionary
```

- **Properties of Dictionary Keys**
- There are two important points to remember about dictionary keys-

(a)More than one entry per key is not allowed. This means no duplicate key is allowed. When duplicate keys are encountered during assignment, the last assignment wins. For example-

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}
print ("dict['Name']: ", dict['Name'])
```

Python Dictionary

- **(b)** Keys must be immutable. This means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example-
- `dict = {['Name']: 'Zara', 'Age': 7}`

```
# TypeError: unhashable type: 'list'
```

- Examples:

```
dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
print (dict['one']) # Prints value for 'one' key
print (dict[2]) # Prints value for 2 key
print (tinydict) # Prints complete dictionary
print (tinydict.keys()) # Prints all the keys
print (tinydict.values()) # Prints all the values
```

Built-in Dictionary Methods

Methods with Description
dict.clear() Removes all elements of dictionary *dict.*
dict.copy() Returns a shallow copy of dictionary *dict.*
dict.fromkeys() Create a new dictionary with keys from seq and values *set* to *value*.
dict.get(key, default=None) For *key* key, returns value or default if key not in dictionary.

Built-in Dictionary Methods

5	**dict.items()** Returns a list of *dict*'s (key, value) tuple pairs.
6	**dict.keys()** Returns list of dictionary dict's keys.
7	**dict.setdefault(key, default=None)** Similar to get(), but will set dict[key]=default if *key* is not already in dict.
8	**dict.update(dict2)** Adds dictionary *dict2*'s key-values pairs to *dict.*
9	**dict.values()** Returns list of dictionary *dict*'s values.

Python Dictionary Examples

- **Examples**
- The following example shows the usage of fromkeys() method.

```
seq = ('name', 'age', ‘gender')
dict = dict.fromkeys(seq)
print ("New Dictionary : %s" % dict)
dict = dict.fromkeys(seq, 10)
print ("New Dictionary : %s" % dict)
```

- The following example shows the usage of get() method.

```
dict = {'Name': 'Zara', 'Age': 27}
print ("Value : %s" % dict.get('Age’))
print ("Value : %s" % dict.get(‘Address'))
print ("Value : %s" % dict.get(‘gender’, “Ma"))
```

- The following example shows the usage of items() method.

```
dict = {'Name': 'Zara', 'Age': 7}
print ("Value : %s" % dict.items())
```

- The following example shows the usage of keys() method.

```
dict = {'Name': 'Zara', 'Age': 7}
print ("Value : %s" % dict.keys())
```

Python Dictionary Examples

- The following example shows the usage of update() method.
- `dict = {'Name': 'Zara', 'Age': 7}`
- `dict2 = {'Sex': 'female' }`
- `dict.update(dict2)`
- `print ("updated dict : ", dict)`

- The following example shows the usage of values() method.
- `dict = {'Sex': 'female', 'Age': 7, 'Name': 'Zara’}`
- `print ("Values : ", list(dict.values()))`

Sets

- Sets are unordered and unindexed collections of unique objects, They are a lot like lists with no repeats. Sets are denoted by curly braces, like below:
- S = {1,2,3,4,5}
- Recall that curly braces are also used to denote dictionaries, and {} is the empty dictionary. To get the empty set, use the **set** function with no arguments, like this:
- S = **set**()
- This **set** function can also be used to convert things to sets. Here are two examples:
- **set**([1,4,4,4,5,1,2,1,3])
- **set**('this is a test')
- Output:

```
{1, 2, 3, 4, 5}
{'a', ' ', 'e', 'i', 'h', 's', 't'}
```

Sets

- Notice that Python will store the data in a set in whatever order it wants to, not necessarily the order you specify. It's the data in the set that matters, not the order of the data. This means that indexing has no meaning for sets. You can't do s[0], for instance.

Working with sets There are a few operators that work with sets.

Operator	Description	Example
|	union	{1,2,3} | {3,4} → {1,2,3,4}
&	intersection	{1,2,3} & {3,4} → {3}
-	difference	{1,2,3} - {3,4} → {1,2}
^	symmetric difference	{1,2,3} ^ {3,4} → {1,2,4}
in	is an element of	3 in {1,2,3} → True

Sets

- The symmetric difference of two sets gives the elements that are in one or the other set, but not both. Here are some useful methods:

Method	Description
`S.add(x)`	Add x to the set
`S.remove(x)`	Remove x from the set
`S.issubset(A)`	Returns `True` if $S \subset A$ and `False` otherwise.
`S.issuperset(A)`	Returns `True` if $A \subset S$ and `False` otherwise.

- **Example1: removing repeated elements from lists?** We can use the fact that a set can have no repeats to remove all repeats from a list.
- L = [1,4,4,4,5,1,2,1,3]
- L = **list**(**set**(L))

Sets cont.

- There are two types of set:

1. Sets - They are mutable and new elements can be added once sets are defined

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket)                    # duplicates will be removed
> {'orange', 'banana', 'pear', 'apple'}
a = set('abracadabra')
print(a)                         # unique letters in a
> {'a', 'r', 'b', 'c', 'd'}
a.add('z')
print(a)
> {'a', 'c', 'r', 'b', 'z', 'd'}
```

2. Frozen Sets - They are immutable and new elements cannot added after its defined.

```
b = frozenset('asdfagsa')
print(b)
> frozenset({'f', 'g', 'd', 'a', 's'})
cities = frozenset(["Frankfurt", "Basel","Freiburg"])
print(cities)
> frozenset({'Frankfurt', 'Basel', 'Freiburg'})
```

Data Type Conversion

- Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type-name as a function.
- There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

Function	Description
int(x [,base])	Converts x to an integer. The base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.

Function	Description
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.
oct(x)	Converts an integer to an octal string.

Input function

- Similar to the print function
- There is an input function()
  - It always reads the input as a string
- 2 ways
  - input() which reads directly
  - input(msg) which prints a message first
- Notice That:
- Input() function reads a complete line, and return as string
- Let's say you want to read 2 numbers
  - Then don't enter both of them on the same line
  - Use 2 input() and input twice (2 lines)
  - a = input()
  - b = input()

More convenient way

- **Input numbers:**

```
>>> var = int ( input () )
33
>>> print ( var + 1 )
34
```

- الإدخال : نستخدم الأمر input لطلب الإدخال من المستخدم.
- إدخال الأعداد : نستخدم الأمر int أو float مع الأمر input لإدخال الأعداد.

- Let's say you want to read 3 strings from a single line, you can use this line of code (later you will understand)
  - **a, b, c = input().split()**
  - **Then: a, b, c are 3 strings**
- Let's read 4 integers:
  - **a, b, c, d = map(int, input().split())**
  - **But you must enter really 4 integers**
- Let's read 5 floats
  - **a, b, c, d, e = map(float, input('Enter 5 numbers: ').split())**
- Follow this syntax style for now