ALGORITHMIC THINKING WITH PYTHON

MODULE - 1

Part III

Module – 1 Part -III

ESSENTIALS OF PYTHON PROGRAMMING: -

Creating and using variables in Python,

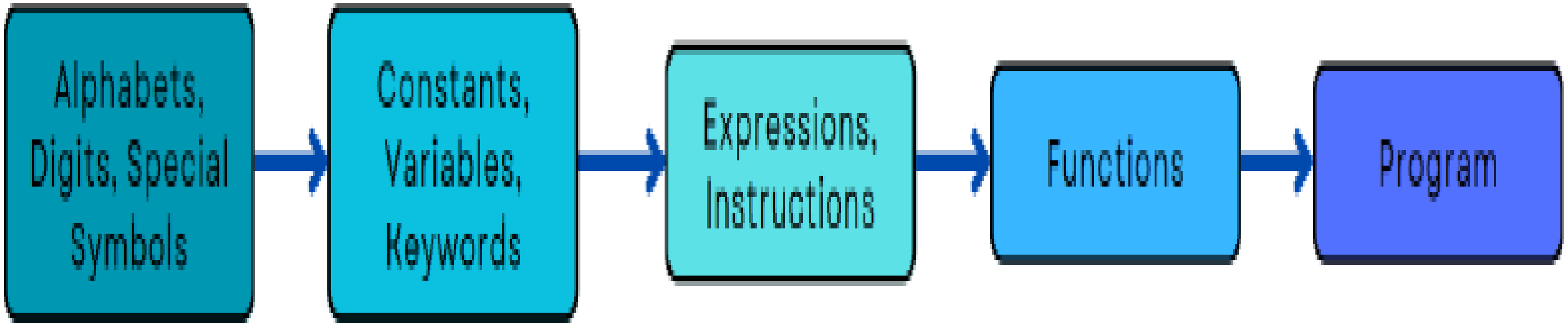
Numeric and String data types in Python,

Using the math module,

Using the Python Standard Library for handling basic I/O - print, input,

Python operators and their precedence.

Steps in learning a programming language



Alphabets, Digits, Special Symbols

Constants, Variables, Keywords

Expressions, Instructions

Functions

Program

Character set

The set of characters supported by a programming language is called **character set**.

A character can be an alphabet, a digit, or a special symbol.

Python supports the following characters:

✓ Upper case alphabets (A–Z)

✓ Lower case alphabets (a–z)

✓ Digits (0–9)

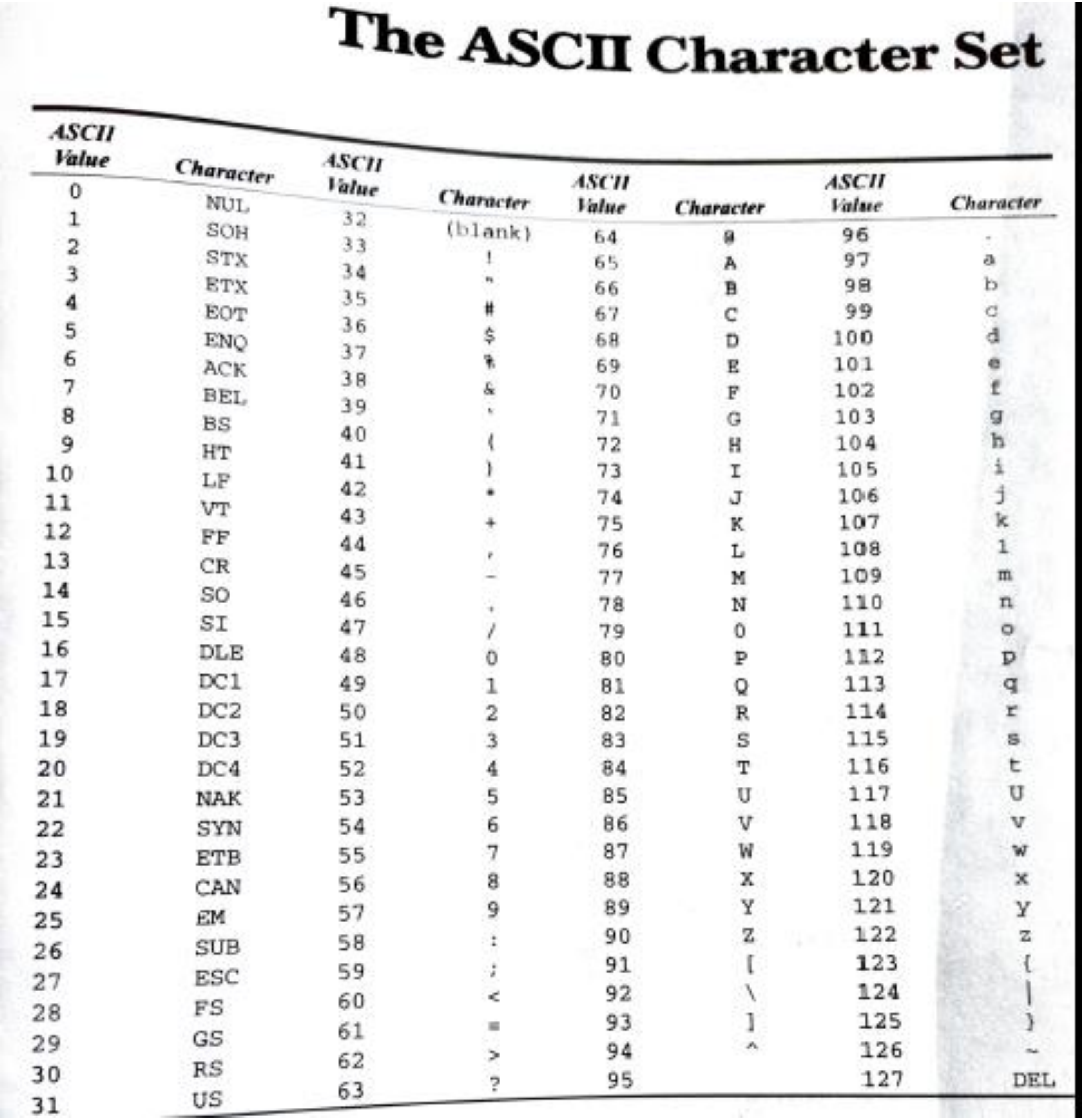
✓ Special symbols like @,#,%,$ etc.

Special Character	Description
, (comma)	{ (opening curly bracket)
. (period)	} (closing curly bracket)
; (semi-colon)	[(left bracket)
: (colon)	] (right bracket)
? (question mark)	((opening left parenthesis)
‘ (apostrophe)	) (closing right parenthesis)
” (double quotation mark)	& (ampersand)
! (exclamation mark)	^ (caret)
| (vertical bar)	+ (addition)
/ (forward slash)	– (subtraction)
\ (backward slash)	* (multiplication)
~ (tilde)	/ (division)
_ (underscore)	> (greater than or closing angle bracket)
$ (dollar sign)	< (less than or opening angle bracket)
% (percentage sign)	# (hash sign)

ASCII

American Standard Code for Information Interchange:

- a standard data-encoding format for electronic communication between computers. ASCII assigns standard numeric values to letters, numerals, punctuation marks, and other characters used in computers.
- This code is basically used for identifying characters and numerals in a keyboard.



The ASCII Character Set

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

ASCII

- Python maps each valid character to an integer value called **ASCII value**.
- For example, the ASCII value of

character **'A'** is **65**

character **‘a'** is **97**

Character	Binary	Decimal	HexaDecimal
A	01**0**0 0001	65	0x41
a	01**1**0 0001	97	0x61

Identifiers

Identifiers are names used to identify variables, functions, classes, modules, and other objects in Python.

Rules for identifiers

- It cannot be a reserved python keyword.
- It should not contain white space.
- It can be a combination of A-Z, a-z, 0-9, or underscore.
- It should start with an alphabet character or an underscore (_).
- It should not contain any special character other than an underscore (_).

Constants

- A constant is a value that never changes during processing or execution of program.
- It can be of any datatype

Example

```
pi = 3.14
g = 9.8
```

- Python programmers use all capital letters

```
MAX_CONNECTIONS = 5000
```

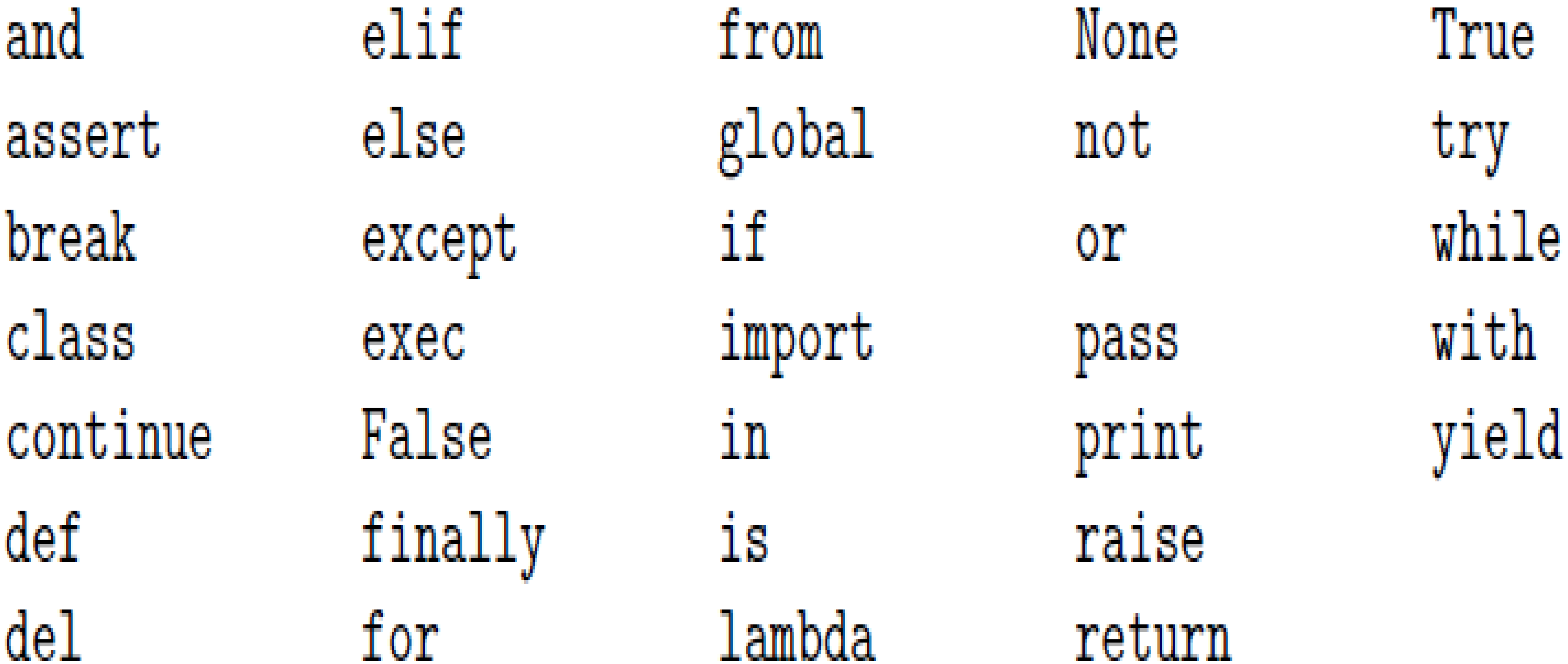
Variable

Variables are names that can be assigned a value

- Variable names must start with a letter or the underscore '_' and can be followed by any number of letters, digits, or underscores.
- Variable names are case sensitive; the variable **Sum** is a different name from the **sum**.
- **Spaces** are not allowed in variable names.
- Variable names cannot be a keyword or a function name.

Keywords

Keywords (also called **reserved words**) are special words, reserved for other purposes; thus, they cannot be used as variable names.

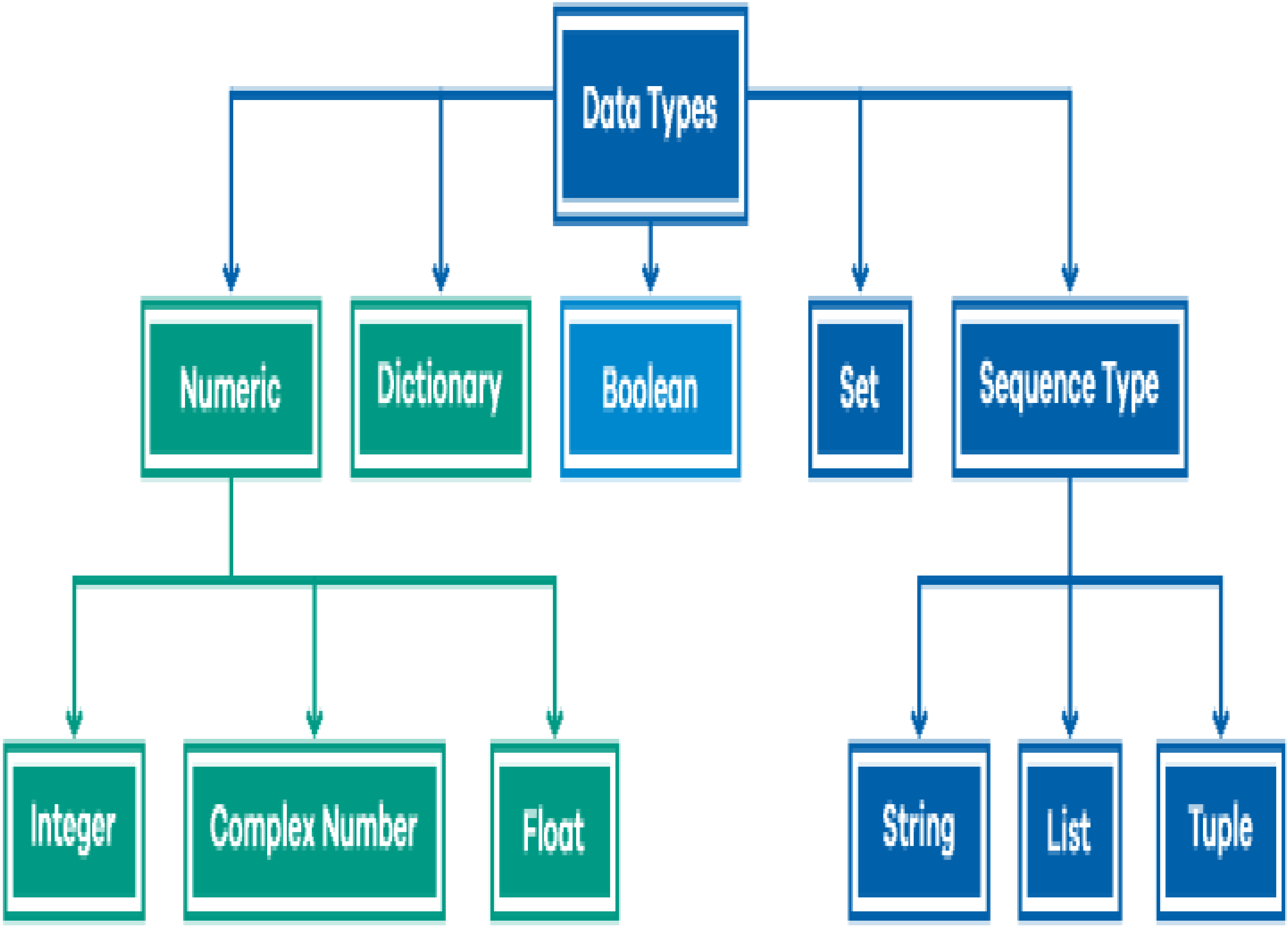


and
assert
break
class
continue
def
del
elif
else
except
exec
False
finally
for
from
global
if
import
in
is
lambda
None
not
or
pass
print
raise
return
True
try
while
with
yield

```
import keyword
print(keyword.kwlist)
```

Variable names

Invalid Identifier ✗	**Valid Identifier** ✓
Total marks	Total_marks
CN	ClientName
Pay-Rate	Pay_Rate
6%salestax	SixPercent_SalesTax
account@Kochi	account_Kochi
return	keywords – not to be used


Data Types
Numeric
Dictionary
Boolean
Set
Sequence Type
Integer
Complex Number
Float
String
List
Tuple

Numeric datatype

- Numbers are used quite often in programming to keep score in games, represent data in visualizations, store information in web applications, and so on.

1. **Integer:** represents whole numbers – both +ve and –ve,

   - Can be of any length (size)

   Example: 5, 3, 0, -100, 987654321

Numeric datatype

2. Float: Python calls any number with a decimal point a *float*.

- It refers to the fact that a decimal point can appear at any position in a number.

Example: 3.14, -0.000001, 149.99

3. Complex: numbers of form a + bj

a - real part,   b - Imaginary part

Example: 5+6**j**,   7.8 + 6.2**j**

Example of datatypes

```
a = 2
b = 3.4
c = 5 + 6j
d = “789"
```

code	output
`type(a)`	int
`type(b)`	float
`type(c)`	complex
`type(d)`	str

float datatype

A floating point number can be written using either ordinary decimal notation or scientific notation.

Scientific notation is often useful for denoting numbers with very large or very small magnitudes.

Decimal notation	Scientific notation	Meaning
3.146	3.146e0	3.146×10^{0}
314.6	3.146e2	3.146×10^{2}
0.3146	3.146e-1	3.146×10^{-1}
0.003146	3.146e-3	3.146×10^{-3}

Sequence datatype

It helps to organise and store multiple values efficiently .

- String, list and tuple

1. String: A *string* is a series of characters enclosed within single or double quotes.

   Example:

   ‘ I told my friend, “Python is my favorite language! ” ’

   “ Good Morning !”

Sequence datatype

2. List: Ordered and mutable collection of items.

- It can store any data type values. Many procedures can be performed in the list, such as append, remove, insert, etc.
- Items are separated by comma and enclosed within [ ]

Example:

First_list = [‘abcd’, 3.14, 25, 99.3, ‘Tom’]

Sequence datatype

3. Tuple: Ordered and immutable collection of items.

- It can store any data type values or components.
- Items are separated by comma and enclosed within ( )

Example:

Small_tuple = (‘Jim’, ‘ marks’, 98, ‘apple’)

Boolean Datatype

- Booleans only have two possible values: True or False.
- True and False – represented as 1 and 0 internally
- Useful in in decision-making process within the code.
- The purpose of boolean values is to represent binary test conditions and decisions in a program.
- Example:

```
x = True
type(x)
```

Output:
bool

Mapping

- Mapping is an unordered data type in Python. Currently, there is only one standard mapping data type in Python called Dictionary.
- Dictionary in Python holds data items in key-value pairs and items are enclosed in curly brackets { }
- Every key is separated from its value using a colon ':' sign.
- The key value pairs of a dictionary can be accessed using the key.
- Keys are usually of string type and their values can be of any data type.
- In order to access any value in the dictionary, we have to specify its key in square brackets [ ].

Dictionary- example

```
stud = {'Name': 'Rosy', 'Age': 18, 'Class': 'S1'}
print(stud)
print(stud.keys())
print(stud.values())
```

Output:

```
{'Name': 'Rosy', 'Age': 18, 'Class': 'S1'}
dict_keys(['Name', 'Age', 'Class'])
dict_values(['Ram', 18, 'S1'])
```

Set

- A set is a collection of unique elements that are unordered and mutable.
- Sets are created by placing elements inside curly braces, separated by commas.
- Example

```
        Set1 = {1, 2.4, 3}.
    x = {"apple", "banana", "cherry"}
```

- If there are duplicates in the Set, they are automatically removed.

Set – Example

code	Output

x = {"apple", "banana", "cherry"}
print(x)
	{'banana', 'apple', 'cherry'}

set1 = {1,2,3}
print(set1)
	{1,2 3}

my_set = {1,2,3,2,1}
print(my_set)
	{1,2,3}

num=set([1,2,3,4]) # set from list
print(num)
{1, 2, 3, 4}
	{1, 2, 3, 4}

vowels=set(('a','e','i','o,','u')) print(vowels) # set from tuple	{'o,', 'a', 'e', 'u', 'i'}

Comments

# A *comment* allows to write notes in the spoken language, within programs.

# As programs become longer and more complicated, notes that describe the overall approach can be added to programs

# They are lines of text that don’t affect the way a program runs. They document what code does or why the programmer made certain decisions.

Example:

```
# Say hello to everyone.
# This is my first program
sum = 5 + 7       # the variable sum contains sum of 5 and 7
```

Multi line comments

```
''' This is a multi-line comment
using triple-quoted strings.
It spans multiple lines. ’’’

"""
Program name: areaRect.py
This program finds the area of a rectangle.
The inputs are two integers representing the
length and breadth of a rectangle, and the output
is an integer named area that represents the area
of the rectangle
"""
```

Practice

```
My_string = ‘Welcome to MEC’
tiny_string = “Hello!!”
```

Char	W	e	l	c	o	m	e		t	o		M	E	C
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Code	Output
`print(My_string[5])`	m
`print(My_string[11:14])`	MEC
`print(My_string[8:])`	to MEC
`print(My_string + “   Hostel”)`	Welcome to MEC Hostel
`print(tiny_string_2*2)`	Hello!!Hello!!

Math module

- Python provides many functions ranging from input/output to performing complex calculations.
- Python groups together functions providing similar functionalities into a **module** for easy access to them.

Example:

math, sys, os

Math module

Function	Description
`ceil(x)`	Returns the smallest integer greater than or equal to x.
`log10(x)`	Returns the base-10 logarithm of x
`abs(x)`	Returns the absolute value of x
`factorial(x)`	Returns the factorial of x
`floor(x)`	Returns the largest integer less than or equal to x
`mod(x, y)`	Returns the remainder when x is divided by y

Math module

`cos(x)`	Returns the cosine of x
`sin(x)`	Returns the sine of x
`tan(x)`	Returns the tangent of x
`pow(x, y)`	Returns x raised to the power y
`degrees(x)`	Converts angle x from radians to degrees
`radians(x)`	Converts angle x from degrees to radians
`sqrt(x)`	Returns the square root of x

Math module

Constant	Description
`math.e`	Returns Euler's number (2.7182...)
`math.nan`	Returns a floating-point NaN (Not a Number) value
`math.pi`	Returns PI (3.1415...)

Practice (math module)

Code	Output	Code	Output

`import math`
`x = 9`
`y = math.sqrt(x)`
`print(y)`
	3	**`import math as mt`**
		`x = 5`
		`n = 2`
		`y = mt.pow(x,n)`
		`print(y)`
			25

`from math import sqrt`
`x = 9`
`y = math.sqrt(x)`
`print(y)`
	3	**`import math as mt`**
		`z = mt.sin(0)`
		`print(z)`
			0

`from math import *`
`print(log10(1))`
	0	**`import math as mt`**
		`z = mt.sin((mt.pi)/2)`
		`print(z)`
			1

Basic Input/Output : input, **print**

Code	Output
print(“Hello World!”)	Hello World!
print(“Hello **\t** World!!”	Hello World!
print(“Hello **\n** World!!”	Hello World!
print(“Hello **\v** World!!” ?!! Does it exist now	Hello World!
print("Hello**\b** World!")	Hell World!

\t – Horizontal tab, \n – New line, \b - Backspace

Basic Input/Output : input, **print**

Code	**Output**
print('It\'s my bag')	It's my bag
print("The \"quotes\" in Poem")	The "quotes" in Poem
print("A backward slash is : \\")	A backward slash is : \

\t – Horizontal tab, \n – New line, \b - Backspace

Basic Input/Output : **input**, print

- To get some input (text or number) from a user, use interactive window with **`input( )`**

Example:

```
input()
```

It will prompt the user or wait for the user - to enter any text or number

```
>>> Hai
```

Output : ‘Hai’

input( )

```
Client_name = input("Enter the name of Client : ")
print("The entered name is : ",Client_name)
```

```
>>>    Enter the name of Client : TOM
```

Output: The entered name is : TOM

```
Stud_mark = input("Enter the mark of Student : ")
print("The mark is : ", Stud_mark)
```

```
>>>    Enter the mark of Student :  87
```

Output : The mark is : 87

Practice (input())

```
a = input("Enter First_num : ")
b = input("Enter Second_num : ")
sum = a + b
print(“sum = “ ,sum)
```

```
>>> Enter First_num : 2
>>> Enter Second_num : 3
```

Output:

```
sum  = 23
```

??!!!

Practice (input())

```
a = int(input("Enter First_num : "))
b = int(input("Enter Second_num : "))
sum = a + b
print(“sum = “ ,sum)
```

```
>>> Enter First_num : 2
>>> Enter Second_num : 3
```

```
Output:
sum = 5
```

Operators

Python operators are used to perform specific operations on one or more operands.

The variables, values, or expressions can be used as operands

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

Arithmetic Operators

They are used to perform various arithmetic operations

Operator	Meaning	Example	Result
+	Addition	6 + 3	9
-	Subtraction	8 - 1	7
*	Multiplication	2 * 4	8
/	Division	20/3	6.6667
%	Modulus	5%3	2 (Remainder)
**	Exponent	5**2	25
//	Integer division / floor division	20//3	6

Comparison Operators

They are used to compare the values on either side of the operator

The result of a comparison is either **True** or **False**.

<	Less than	a<b
>	Greater than	a>b
<=	Less than or equal to	a<=b
>=	Greater than or equal to	a>=b
==	is equal to	a==b
!=	is not equal to	a!=b

```
i,j,k = 3,4,7
i > j               #Output : False
(j+k)>(i+5)         #Output : True
```

Assignment Operators

Assignment operators are used to assign values to variables

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

Logical operators

- Python includes three Boolean (logical) operators *viz.* `and` , `or` , and `not` .
- The `and` operator and `or` operator expect two operands

Operator	Description	Example
`and`	Logical AND, Returns **True** if **both** statements are true	x > 5 and x < 10
`or`	Logical OR, Returns **True** if **one** of the statements is true	Phy > 80 or Che > 90
`not`	Logical NOT, Reverse the result, returns False if the result is true	not(4 < 5)

Practice (Arithmetic)

```
a = 10; b = 5; c = 2;
print(‘Sum = ’, (a+b))
print(‘Diff = ’, (a-b))
print(‘Product = ’, (a*b))
print(‘Quotient = ’, (a/b))
print(‘Remainder = ’, (b%c))
print(‘Exponent = ’, (b**2))
print(‘Floor division = ’, (b//c))
```

Practice (relational)

```
a = 10; b = 5;
print(‘a==b is :  ’ , (a==b))
print(‘a!=b is :  ’ , (a!=b))
print(‘a>b is :  ’ , (a>b))
print(‘a<b is :  ’ , (a<b))
print(‘a>=b is :  ’ , (a>=b))
print(‘a<=b is :  ’ , (a<=b))
```

Practice (assignment)

a = 10; b = 5;
a += b
print(a)
	a = 10; b = 5;
	a -= b
	print(a)
		a = 10; b = 5;
		a *= b
		print(a)

a = 10; b = 5;
a /= b
print(a)
	b = 5; c = 2;
	b%=c
	print(b)
		b = 5; c = 2;
		b**=c
		print(b)

b = 5; c = 2;
b//=c
print(b)
	a = 6; b = 3;
	a += b
	print(a)
		a = 6; b = 3;
		a += b
		print(a)

Practice (Logical)

```
x = 10; y = 20;

print("x > 0 and x < 10:",x > 0 and x < 10)

print("x > 0 and y > 10:",x > 0 and y > 10)

print("x > 10 or y > 10:",x > 10 or y > 10)

print("x%2 == 0 and y%2 == 0:",x%2 == 0 and y%2 == 0)

print ("not (x+y>15):", not (x+y)>15)
```

Decimal – Binary conversion

Decimal	Binary	Decimal	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Bitwise operators

- Bitwise operators take the binary representation of the operands and work on their bits, one bit at a time.
- The bits of the operand(s) are compared starting with the rightmost bit – LSB, then moving towards the MSB.

Bitwise operators

Operator	Description
& Binary AND	The operator sends the bit present in both operands to the output.
| Binary OR	If a bit is present in either operand, it is copied.
^ Binary XOR	It is copied if the bit is set inside one argument but not both.
~ Binary Ones Complement	It has the function of 'flipping' bits and is unary.
<< Binary Left Shift	The left operand's value is moved to its left by the number of bits specified in the right argument.
>> Binary Right Shift	The quantity of bits provided by the right parameter advances the position of the left operand.

Bitwise operators

<table>
<tr><th colspan="2" rowspan="2">Input</th><th colspan="3">Output</th></tr>
<tr><th>AND</th><th>OR</th><th>XOR</th></tr>
<tr><th>bit1</th><th>bit2</th><th>bit1 & bit2</th><th>bit1 | bit2</th><th>bit1 ^ b2</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
</table>

Bitwise operators

AND

$$\begin{array}{rl} \mathtt{a} & = 0001\ \ 0100 \\ \mathtt{b} & = 0110\ \ 1100 \\ \hline \mathtt{a\ \&\ b} & = 0000\ \ 0100 \\ & = 4 \end{array}$$

OR

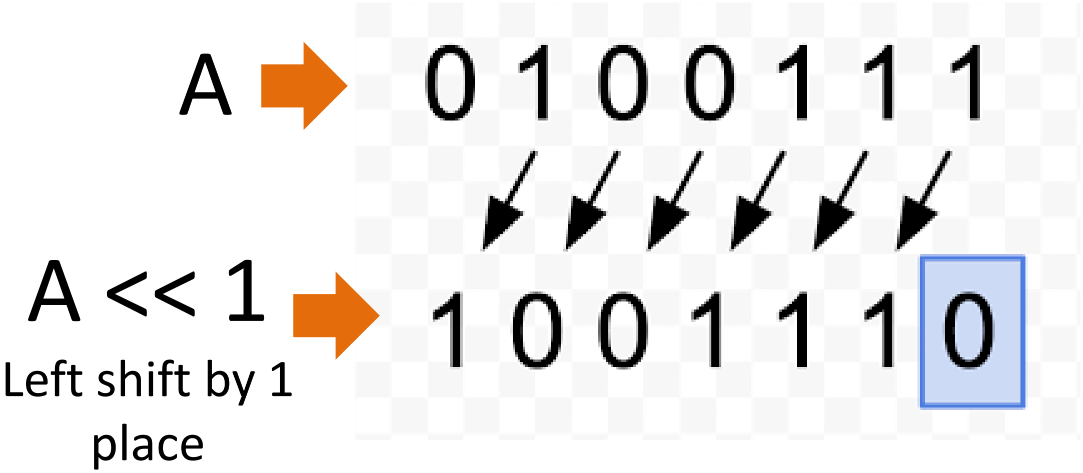
$$\begin{array}{rl} \mathtt{a} & = 0001\ \ 0100 \\ \mathtt{b} & = 0110\ \ 1100 \\ \hline \mathtt{a} \mid \mathtt{b} & = 0111\ \ 1100 \\ & = 124 \end{array}$$

XOR

$$\begin{array}{rl} \mathtt{a} & = 0001\ \ 0100 \\ \mathtt{b} & = 0110\ \ 1100 \\ \hline \mathtt{a}\ \hat{}\ \mathtt{b} & = 0111\ \ 1000 \\ & = 120 \end{array}$$

Bitwise left shift <<

- When we perform a left shift on binary number, we will move the bits to the left, adding a new bit to the right of the binary.
- This new bit will always be 0 for this operation, resulting in a new binary.



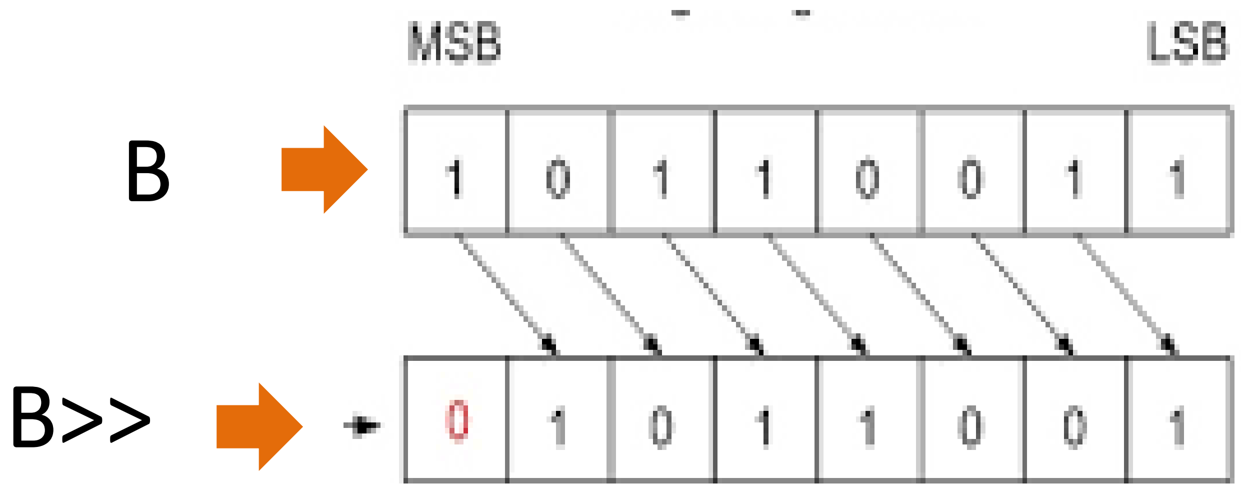
A 0 1 0 0 1 1 1

A << 1 1 0 0 1 1 1 0

Left shift by 1 place

Bitwise right shift >>

- We will move the bits to the right. Thus, the bit farthest to the right of the old binary will be removed from the new binary.



	MSB							LSB
B	1	0	1	1	0	0	1	1
B>>	0	1	0	1	1	0	0	1

Practice (Bitwise)

a = 60; b = 2; (60)D = (0011 1100)b (2)D = (0000 0010)b	**Output**
print(a&b)	0
print(a|b)	62
print(a^b)	62
print(~a)	-61
print(a>>b)	15
print(a<<b)	240

Membership operators

- These operators test for the membership of a data item in a sequence, such as a string.

Operator	Description
`in`	By using the in operator, one can determine if a value is present in a sequence or not. If the specified variable/literal is found, it will return **True**, otherwise, it will return **False**.
`not in`	By using the, not in operator, one can determine if a value is not present in a sequence or not. If the specified variable/literal is found, it will return **False**, otherwise it will return **True**.

Membership operators

```
mrk = [10,20,30,40]
a = 20; b = 10; c = a-b; d = a/2;
print (a, "in", mrk, ":", a in mrk)
print (b, "not in", mrk, ":", b not in mrk)
print (c, "in", mrk, ":", c in mrk)
print (d, "not in", mrk, ":", d not in mrk)
```

Output

```
20 in [10, 20, 30, 40] : True
10 not in [10, 20, 30, 40] : False
10 in [10, 20, 30, 40] : True
10.0 not in [10, 20, 30, 40] : False
```

```
'A' in 'ASCII’
          # Output : True
'a' in 'ASCII’
          # Output : False
'a' not in 'ASCII'
          # Output : True
```

Identity Operator

- Used to check whether two variables point to the **same object** in memory

Operator	Description	Example
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y

Identity Operator

```
x = ["bread", "cake"]
y = ["bread", "cake"]
z = x
print(x is z)
print(x is not y)
```

```
Output:
True
True
```

True, Since this returns True as x has been assigned to z, therefore, z and x points to the same object.

True, Since this returns True as x and y are not the same objects even though they have the same content.

Operator Precedence

Precedence	Operators	Description
1	()	Parentheses
2	**	Exponentiation
3	~, + , -	Complement, unary plus, minus
4	*, /, //, %	Multiplication, matrix, division, floor division, remainder
5	+, -	Addition and subtraction
6	<<, >>	Shifts
7	&	Bitwise AND
8	^	Bitwise XOR
9	|	Bitwise OR
10	in,not in, is, is not, <, <=, >, >= , !=, ==	Comparisons, membership tests, identity tests
11	not, or , and	Logical operator

Operator precedence

```
a = 20; b = 10; c = 15; d = 5; e = 0;
e = (a + b) * c / d #( 30 * 15 ) / 5
print ("Value of (a + b) * c / d is ", e)
e = ((a + b) * c) / d # (30 * 15 ) / 5
print ("Value of ((a + b) * c) / d is ", e)
e = (a + b) * (c / d); # (30) * (15/5)
print ("Value of (a + b) * (c / d) is ", e)
e = a + (b * c) / d; # 20 + (150/5)
print ("Value of a + (b * c) / d is ", e)
```

Output:

```
Value of (a + b) * c / d is 90.0
Value of ((a + b) * c) / d is 90.0
Value of (a + b) * (c / d) is 90.0
Value of a + (b * c) / d is 50.0
```

Operator precedence

How will Python evaluate the following expression?

20 + 30 * 40

Solution:

#precedence of * is more than that of +

= 20 + 1200 #Step 1

= 1220 #Step 2

Operator precedence

How will the following expression be evaluated?

15.0 / 4.0 + (8 + 3.0)

Solution:

= 15.0 / 4.0 + (8.0 + 3.0) #Step 1

= 15.0 / 4.0 + 11.0 #Step 2

= 3.75 + 11.0 #Step 3

= 14.75 #Step 4

Table 6.2: Escape sequences in Python

Escape Sequence	**Meaning**
\b	Backspace
\n	Newline
\t	Horizontal tab
\v	Vertical tab
\\	The \ character
\'	Single quotation mark
\"	Double quotation mark