Unit - 1

Page No.
Date

Introduction :-

- Python programming language was developed by Guido Van Rossum in February 1991 at CWE in neitherland.
- python is based on or influenced with two programming languages:
  - ⇒ ABC language
  - ⇒ Modula- 3

- python is a high level programming lang.
- Python is an open source programming lang.
- Object oriented programming lang.
- It is an interpreted lang.
- It is a scripting lang.
- No Need to compile.
- Automatic Garbage collectim
- Dynamic type checking.

Use :- web programming, scripting, scientic Computing AI

Version of pythm :- 1.X, 2.X, 3.X

Python - Pluses :-

1. Easy to use :- pythm is compact and very easy to use Object oriented lang with very simple syntax rules.

2. Expressive language :-

python's expressiveness means it is more capable to express the code's purpose than many other languages.

Reason being - fewer lines of code, simple syn

// In C++ : Swap values

```
int a = 2, b = 3, tmp;
tmp = a;
a = b;
b = tmp;
```

In python : Swap Valu

```
a, b = 2, 3
a, b = b, a
```

3. Interpreted Language :-

Python is an interpreted language, not a compiled language.

This means that the python installatin interprets and executes the code line by line at a time.

4. Its completeness :-

- when we install pythn, we get everything we need to do real work.
- we do not need to download and install additional libraries ;

5. Crose - platform Language :-

python can run equally well on variety of platforms - windows, Linux / Unix, Macintosh, supercomputers, smart phones. In other words, pythn is a portable language.

Page No.
Date

Python - Some Minuses

- Although Python is very powerful yet simple language with so many advantages, it is not the Perfect Programming Language.

1. Not the fastest language
   - python is an interpreted language not a fully compiled one.
   - python is first semi-compiled into an internal byte-code, which is then exerted by a python interpreter.
   - So, here python is little weaker though it offers faster development times but execution-times are not that fast compared to some compiled languages.

2. Lesser Libraries than C, Java, Perl
   python offers library support for all computing programs, but its library is still not competent with languages like C, Java, Perl as they have larger collections available.

3. Not Strong on Type-binding :-
   python interpreter is not very strong on 'type-mismatch' issues.
   For example, if you declare a variable as integer but later store a string value in it python won't complain or pin-point it.

Page No.
Date

Working in python :-

Before we start working in python, we need to install Python on our computers. There are multiple python distributions available today.

⇒ Default installation available from www.python.org is called CPython installation and comes with python interpreter, python IDLE and pip (package installer).

⇒ There are many python distributions available these days. Anaconda Python distribution is one such highly recommended distribution that comes preloaded with many packages and libraries (e.g. Numpy, Scipy, Panda libraries etc.)

⇒ Many popular IDEs are also available e.g., Spyder IDE, pytcharm IDE etc.

⇒ we can work in python in following different ways:

(i) in Interactive mode (also called Immediate Mode)
(ii) in Script mode.

* working in Interactive Mode (Python IDLE)

Interactive mode of working means you type the command - one command at a time, and the python executes the given command there and then gives you output.

In interactive mode, you type the command in front of Python command prompt >>>.

>>> 2+5 ← Command/expression given here

Result → 7
Returned by
python

To work in interactive mode, follow the process given below:

(i) Click Start button → All programs → Python 3.6.X → IDLE (Python GUI)

→ print("Hello") → Hello
print('Hello')

→ "Hello" → 'Hello'

→ >>> S = 13
>>> S
13

→ >>> S+20
33

* Working in Script mode →
With interactive mode, you cannot do or save all the commands in the form of program file and can not see all the output lines together. So the solution for this problem is the script mode.

To work in a script mode, we need to do the following:

Step 1: Create Module / Script / Program file

(i) Click Start button → All programs → Python 3.6.x → IDLE

(ii) Click File → New in IDLE Python Shell.

(iii) In the New window that opens, type the commands you want to save in the form of a program (or script).

```
print("Hello World!")
```

(iv) Click File → Save and then save the file with an extension .py.

Step 2:- Run Module / Script / Program file

(i) Open the desired program / script file that we created in previous step 1 by using IDLE's file → open command.

(ii) Click Run → Run Module command in the open program / script file's window.
we may also press F5 key.

(iii) It will execute all the commands stored in module / program / script that we had opened and show the complete output in a separate python Shell window

$(8)_{10} = (\ )_8$

Page No. 8 8
Date 1-0

The integers

The numeric literals in python can belong to any of the following four different numerical types:

int (signed integers) :→

. Integer literals are whole numbers without any fractional part. The method of writing integer constants has been specified in the following rule.

. An integer constant must have atleast one digit and must not contain any decimal point.

Python allows three types of integer literals:

(i) Decimal Integer Literals :→

An integer literal consisting of a sequence of digits is taken to be decimal integer literal unless it begins with 0.

1234, 41, +97, -17 are decimal integer literals

(ii) Octal Integer Literals : A sequence of digits starting with 0o (digit zero followed by letter O) is taken to be an Octal integer.

Decimal integer 8 will be written as 0o10 as Octal integer.

(iii) Hexadecimal integer Literals :-

A sequence of digits preceded by 0x or 0X is taken to be an Hexadecimal integer.

Page No.
Date

Floating Point Literals →

Floating literals are also called real literals.

Real literals are numbers having fractional parts. These may be written in one of the two forms called Fractional Form or the Exponent form.

1. Fractional Form: A real literal in Fractional form consist of signed or unsigned digits including a decimal point between digits.

```
>>> 5.0 → 5.0
>>> 5.50 → 5.5
>>> 5.8540000 → 5.854
>>> 5.05 → 5.05

>>> 10/2 → 5.0
>>> 10.0/2 → 5.0
>>> 10.0/2.0 → 5.0
>>> (4+8)/2 → 6.0
>>> -7 → -7
>>> (-7+2) * (-4) → 20
>>> (1+9)/(-5+5) → error
>>> 3/4 → 0.75
>>> 9.8765000 → 9.8765
```

** → exponentiation
// → Quotient or floor divisim
/ → Division
% → Remainder

Page No.
Date

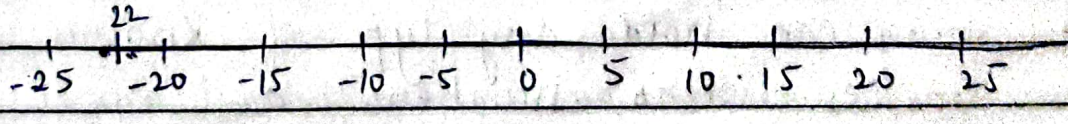
>>> 9 ** (1/2) → 3.0

>>> 20 // 6 = 3
>>> 5/2 ⇒ 2.5
>>> 6/2 → 3.0
>>> 5.0/2 → 2.5
>>> 6.0/2 → 3.0
>>> 5.0/2.0 → 2.5
>>> 10/3 → 3.3333
>>> 22/5 → 4.4

// → 5//2 → 2
6//2 → 3
5.0//2 → 2.0

% → a, b = Let a & b are two no.
a % b → Result → if b is +ve = result +ve
if b is -ve = -ve

22 % 5 = 2

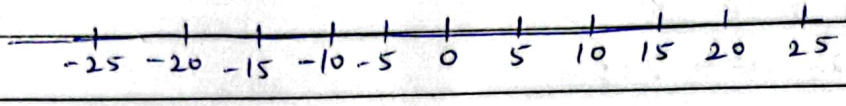


if a is +ve → left
if a is -ve → Right

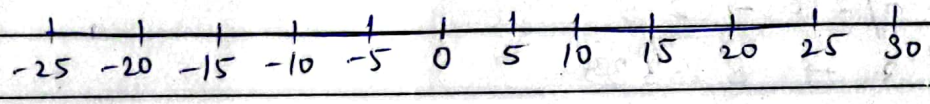
22 % -5 = -3

Page No. _____
Date

$-22 \% 5 = 3$



$-22 \% -5 = -2$



String Data Types :-

A string data type lets you hold string data, i.e any number of valid characters into a set of quotation marks.

- In Python 3.x, each character stored in a string is a unicode character.
- Unicode is a system designed to represent every character from every language.
- A string can hold any type of known characters i.e, letters, numbers, and special characters, of any known scripted language.

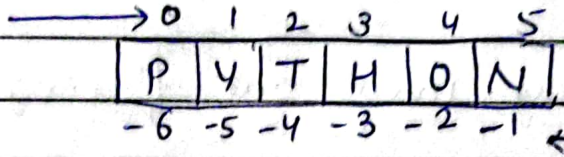
Following are all legal strings in python:
"abcd", "1234", '$%^&', ????

Page No.
Date

String as a Sequence of Characters :-

A Python string is a sequence of characters and each character can be individually accessed using its index.

Forward indexing



name[0] = 'P' = name[-6]
name[1] = 'Y' = name[-5]
name[2] = "T' = name[-4]
name[3] = 'H' = name[-3]
name[4] = 'O' = name[-2]
name[5] = 'N' = name[-1]

Since length of string variable can be determined using function len(<string>),

⇒ first character of the string is at index 0 or - length

⇒ Second character of the string is at index 1 or -(length - 1)

⇒ Second last character of the string is at index (length -2) or -2

⇒ last character of the string is at index (length - 1) or -1.

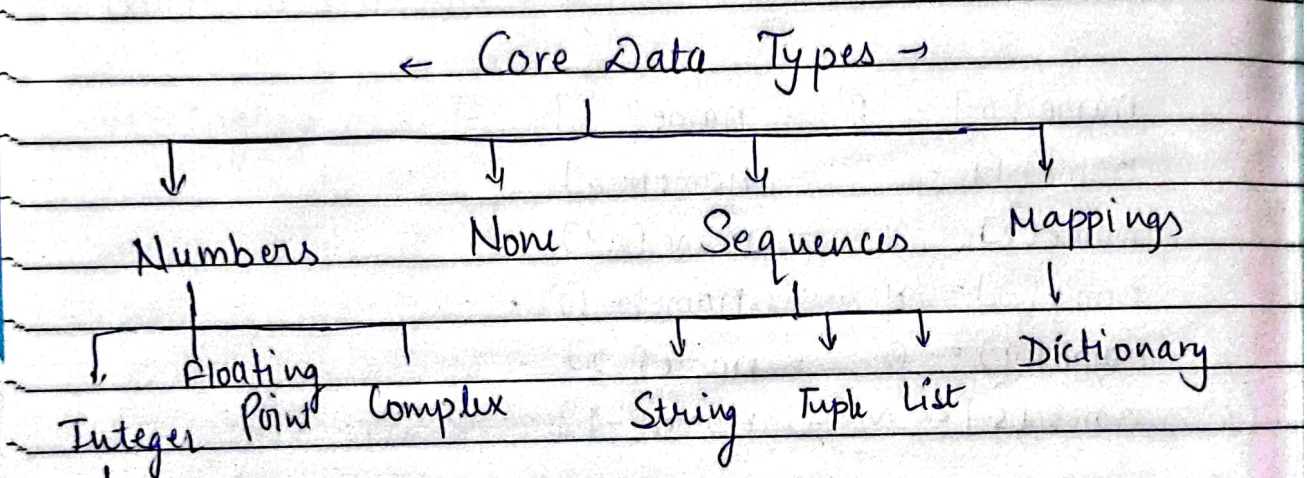
In a string, say name, of length ln, the valid indices are 0, 1, 2, -- ln-1.

That means, if you try to give something like :

>>> name[ln]

python will return an error like :

IndexError : string index out of range



Mutable and immutable Types :→

The python data objects can be broadly categorized into two - mutable and immutable types, in simple words changeable or modifiable and non-modifiable types.

The mutable types are those whose values can be changed in place.

- Lists
- Dictionary
- Sets

Page No.
Date

Immutable Types -

The immutable types are those that can never change their values in place.

In python,

- integers
- floating point numbers
- Boolean
- strings
- Tuples

Operators :-

Arithmetic Operators

+ →	addition	// →	floor division
− →	Subtraction	% →	Remainder
* →	multiplication	** →	exponentiation
/ →	division		

Unary Operators →

The operators unary '+' precedes an operand.

if a = 5 then +a means 5.
if a = 0 then +a means 0.
if a = -4 then +a means -4.

if a = 5 then -a means -5.
if a = 0 then -a means 0
if a = -4 then -a means 4.

Page No. _____
Date

Arithematic

Binary operators :-

1. Additim operator +
2. Subtractim operator -
3. Multiplicatim operator *
4. Divisim /
5. Floor Divisim //
6. Modulus operator %.

Augmented Assignment Operators →

$a = a + b$

we may write

$a += b$

$x += y \Rightarrow x = x + y$

$x -= y \Rightarrow x = x - y$

$x \%= y \Rightarrow x = x * y$

$x /= y \Rightarrow x = x / y$

$x //= y \Rightarrow x = x // y$

$x **= y \Rightarrow x = x ** y$

$x \%= y \Rightarrow x = x \% y$

Relational Operators → Relational refers to the relationship that values can have with one another. Thus, the relational operators determine the relation among different operands.

<	<=	==
>	>=	!=

(i) for numeric types, the values are compared after removing trailing zeros after decimal point from a floating point numbers.

for ex. 4 and 4.0 will be treated as equal (after removing trailing zeros from 4.0, it becomes equal to 4 only).

(ii) Strings are compared on the baris of lexicographical ordering

- Capital letters are considered lesser than small letters, e.g. 'A' is less than 'a'; according to ASCII values.

pythm is not equal to Python

Cod < god will return True

Identity Operators :-

There are two identity operators in python is and is not.

a is b → returns True if both its operands are pointing to same Object

a is not b → returns True if both its operands are pointing to same Object (both references to the same memory locations), returns False otherwise.

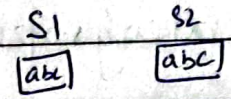
Page No.
Date

is not a is not b returns True if both its operands are pointing to different Object, returns false otherwise.

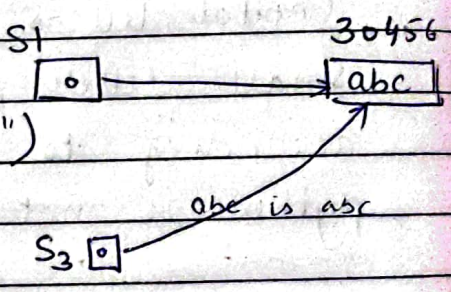
Equality (==) and identity (is)

```
a = 235
print (a, b)
235  235
```

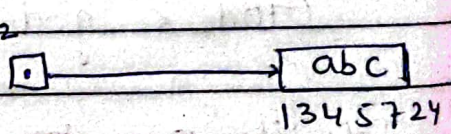
a is b → True
a == b → True



* S1 = 'abc'
S2 = input ("Enter a String:")



↳ Enter a String : abc
S1 == S2 → True



S1 is S2 → False
S3 = 'abc'
S1 is S3 → True

* & i = 2+3.5j
j = 2+3.5j

i is j → False

Also K = 3.5
l = float (input ("Enter a value:"))
Enter a value

K == l
True

K is l
False

The reason behind this behaviour is that there are few cases where python creates two different objects that both store the same value. These are :

⇒ input of strings from the console.
⇒ writing integers literals with many digits
⇒ writing floating-point and complex literals.

Logical Operators →

OR → The OR operator evaluates to True if either of its (relational) operands evaluates to True
AND
NOT

x	y	x or y
F	F	F
F	T	T
T	F	T
T	T	T

(4 == 4) or (5 == 8) → True
5 > 8 or 5 < 2 → False

0 or 0 → 0
0 or 8 → 8
5 or 0.0 → 5
'hello' or '' → 'hello'
'' or 'a' → 'a'
'' or '' → ''
'a' or 'j' → 'a'

The 'and' Operator → It evaluates to 'True' if both of its operands evaluates to True; False if either or both operands evaluate to false

x	y	x and y
F	F	F
F	T	F
T	F	F
T	T	T

(4==4) and (5==8) → False
5>8 and 5<2 → False
8>5 and 2<5 → True

Q. Numbers / strings / lists as operands
when and operator has its operands as numbers or string or lists (e.g 'a' or '', 3 or 0) then the and ~~operator~~ operator performs as per following principle:
In an expression x and y, if first operand has false, val, then return first operand x as result, otherwise return y.

x	y	x and y
f	f	x
f	T	x
T	f	y
T	T	y

0 and 0 → 0
0 and 8 → 0
5 and 0.0 → 0.0
'hello' and '' → ''
'' and 'a' → ''
" and " → "
'a' and 'j' → 'j'

10 > 20 and "a" + 10 < 5
(F) and (wrong exp.)
will give you result as False
ignoring the second operand completely, even if it is wrong.

Note :- The and operator will test second operand only if the first operand is true, otherwise ignore it.

The 'not' operator :-
It works on single expression or operand, i.e it is a unary operator.
The logical not operator negates or reverses the truth values of the expression

* The 'not' operator returns always a Boolean value True or False.

not 5 → False because 5 is non zero
not 0 → True because 0 is false
not -4 → False because -4 is non zero.
not (5>2) → False because 5>2 is True
not (5>9) → results into True because 5>9 is false

Chained Comparison Operators :→

. Rather than writing 1<2 and 2<3 we can write 1<2<3 → True >>> 1<2 and 2<3
True

As per the property of and, the expression 1<3 will be first evaluated and if only it is True, then only the next chained expression 2<3 will be evaluated.

11 < 13 > 12 → True

Bitwise Operators →

Bitwise operators are used to change individual bits in an operand.

Operator	Operation	Use
&	bitwise and	op1 & op2
|	bitwise or	op1 | op2
^	bitwise Xor	op1 ^ op2
~	bitwise complement	~ op1

The AND operator &

op1	op2	Result
0	0	0
0	1	0
1	0	0
1	1	1

13 → 1101 . bin(13) → '0b1101'

12 → 1100 bin(12) → '0b1100'

↳ 13 & 12 → 12

↳ bin(13 & 12) → '0b1100'

The inclusive OR operator :-

op 1	op 2	Result
0	0	0
0	1	1
1	0	1
1	1	1

bin(13) → '0b1101'

bin(12) → '0b1100'

bin(13 | 12) → '0b1101'

13 | 12 → 13

The exclusive OR → ^

if the two operands bits are different, the result is 1; otherwise the result is 0.

The exclusive OR (^) operation

op1	op2	Result
0	0	0
0	1	1
1	0	1
1	1	0

bin(13) → '0b1101'
bin(12) → '0b1100'
1

13 ^ 12 → 1

bin(13 ^ 12) → '0b1'

The Complement Operator → ~ inverts the values of each bit of the operand.

op 1	Result
0	1
1	0

Operator Precedence :-

Operator	Description
()	Parentheses
**	Exponentiation
~x	Bitwise nor
+x, -x	unary
*, /, //, %	Multiply, Division, floor division
+, -	Addition, Subtraction
&	Bitwise and
^	Bitwise XOR
|	Bitwise OR
<, <=, >, >=, <>, !=, ==, is, is not	Comparisons, identity operator

not x	Boolean NOT
and	Boolean AND
or	Boolean OR

Operator Associativity →

- Associativity is the order in which an expression is evaluated.

- Almost all the operators have left-to-right associativity except exponentiation (**), which has right to left associativity.

7 * 8 / 5 // 2 → 5.0

(((7 * 8) / 5) // 2) → 5.0

7 * ((8 / 5) // 2) → 0.0

} left-to right

7 * (8 / (5 // 2)) → 28.0

3 ** 3 ** 2 → 19683

3^{3^2} ←

3 ** (3 ** 2) → 19683

(3 ** 3) ** 2 → 729

Page No.
Date

Python character set →

- Character set is a set of valid characters that a language can recognize.

- python support Unicode encoding standard.

Letters → A-z, a-z
Digits → 0-9
Special Symbols → space + - % * / [ ] //

Tokens → The smallest individual unit a program is known as Token or a lexical unit

python has following tokens:

(i) Keywords
(ii) identifiers
(iii) Literals
(iv) Operators
(v) Punctuators

Keywords :- Keywords are the words that convey a special meaning to the language compiler / interpreter.

There are 33 keywords reserved as.

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

Page No.
Date

Identifiers :- Identifiers are fundamental building blocks of a program and are used as the general terminology for the names given to different parts of the program

Identifier forming rule of python are being specified below :

=> An identifier is an arbitrarily long sequence of letters and digits.

=> The first character must be a letter ; the underscore (_) counts as a letter.

=> Upper and lower-case letters are different. All characters are significant.

=> The digits 0 through 9 can be a part of the identifier except for the first character.

=> identifiers are unlimited in length.

* python is case sensitive as it treats upper and lowercase characters differently.

=> An identifier must not be a Keyword of Python.

=> An identifier cannot contain any special characters except for underscore (_).

The following are some valid identifiers :

Myfile	DATE9_7_77
MYFILE	_DS
_CHK	FILE13

Page No.
Date

Escape Sequences

\\	Backslash (\)
\'	Single quote(')
\"	Double quote(")
\a	ASCII Bel
\b	ASCII Backspace
\f	ASCII formfeed
\n	New line character
\N{name}	Character named name in the Unicode database

Text 1 = 'hello\
world' → 'helloworld'

Str 1 = '''Hello
World.
There I come!!!''' → Hello
world.
There I come!!!

Str1 = '''a\
b\
c'''

len(Str1) = 5

Str 2 = 'a\
b\
c'

len(str2) = 3

Variables and Assignment →

A variable in Python represents named location that refers to a value and whose values can be used and processed during program run.

marks = 70

we assigned the numeric type value to an identifier name and Python created the variable of the type similar to the type of value assigned.

* variables are Not storage Containers in Python.

```
age = 15
age = 20
```

age [15] 20253

age [20] 20253

Memory address did not change of variable age with change in its value.

L values and R values →

lvalue : expressions that can come on the left of an assignment

rvalue : expressions that can come on the rhs of an assignment

• Multiple Assignments →

1. Assigning same value to multiple variables

```
a = b = c = 10
```

All three labels a, b, c will refer to same location with value 10.

Page No.
Date

2. Assigning multiple values to multiple variables

x, y, z = 10, 20, 30

Simple Input and Output →

variable_to_hold_the_value = input("what is your name?")

what is your name ? → 'simar'

* Input() has a property, which always returns a value of String type.

* Python has enclosed both the values in quotes as 'Simar'.

```
>>> input("what is your 'age?") → '16'
>>> age = input("what is your age ?")
16                                    [16]
>>> age+1
Type error : must be str, not int
```

• Variable address can be accessed using id

```
>>> a = 4.
>>> print(4)
4
>>> print(a)
4
```

```
>>> id(4)
30899132
>>> a=4
>>> id(a)
30899132
```

Page No.
Date

Types of Statement →

- Statements are the instructions given to the computer any kind of action, be it data movements.

- Statements form the smallest executable unit within a python program.

- Empty statement
- Compound statement
- Simple statement

1. Empty statement →

The simplest statement is the pass statement i.e a statement which does nothing. In python an empty statement is pass statement. It takes the following

```
pass
```

* wherever python encounters a pass stmt, Python does nothing and moves to next statement in the flow of contd.

2. Simple statement →

Any single executable statement is a simple statement in python.

For example, following is a simple statement in python:

```
name = input("your name")
```

Page No.
Date

3. Compound Statement →

A compound statements represents a group of statements executed as a unit.

< Compound Statement header) :
< indented body containing multiple simple and / or compound statements >

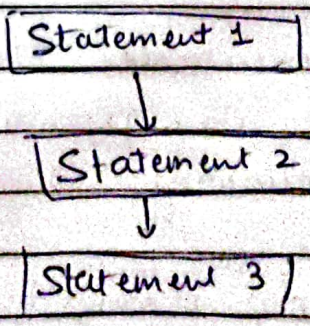
* Header line which begins with a keyword and ends with a colon.

* a body consisting of one or more python statements, each indented insides the header lin.

Statement Flow Control :→

In a program, statements may be executed sequentially, selectively or iteratively. Every programming language provides construct to support sequence, selection or iteration.

Sequence :→ The sequence constructs means the statements are being executed sequentially. This represents the default flow of statement.

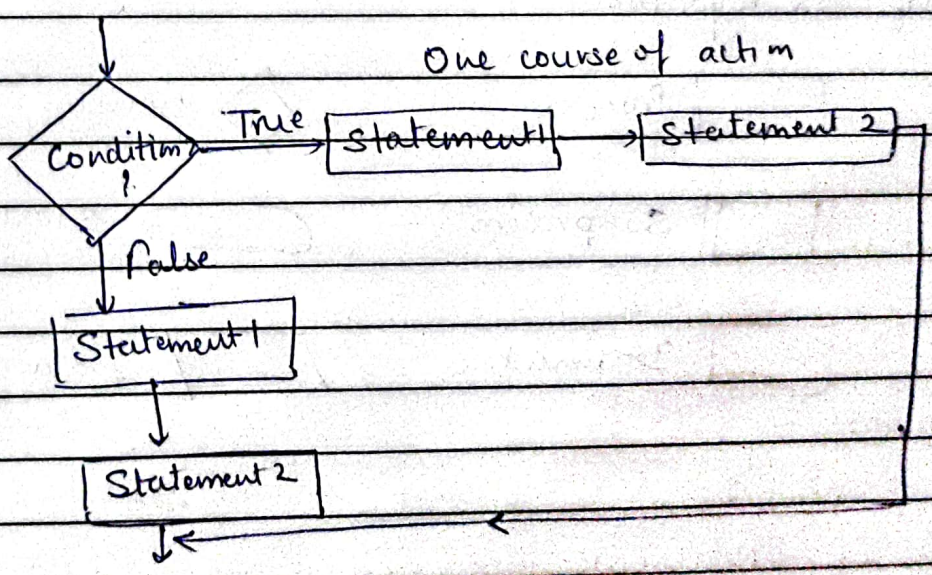


Page No.
Date

Selection - The selection constructs means the execution of statement depending upon a condition-test.

* If a condition evaluates to True, a course of action is followed otherwise another course-of-action is followed.

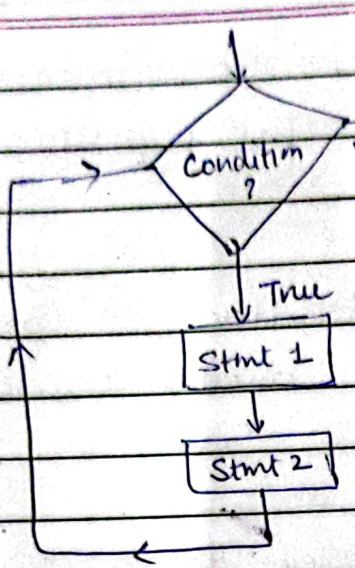
* This construct is also called decision construct because it helps in making decision about which set of statements is to be executed.



Iteration (Looping) :- The iteration constructs means repetition of a set-of-statements depending upon a condition-test.

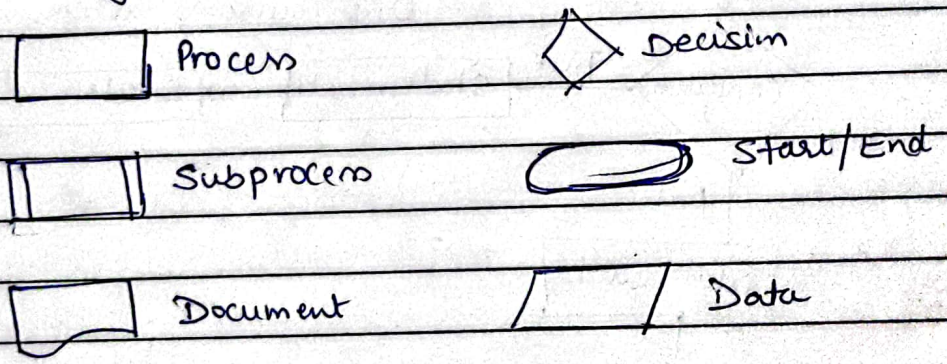
* Till the time a condition is True, a set of statements are repeated again & again.

* As soon as the condition becomes false the repetition stops.



Flowcharts →

A flowchart is a graphical representation of an algorithm.



1) If Statements
2) Looping Statement
3) Jump statements.

1) If statements are conditional statements and implement selection construct.

if < conditional expressions > :
    Statements }

```
ch = input('Enter a single character:')
if ch == ' ':
    print("You entered a space")

if ch >= '0' and ch <= '9':
    print("You entered a digit.")
```

The if-else Statement →

This form tests a condition and if the condition evaluates to True it carried out statements indented below.

```
if < conditional expression > :
    statement
else :
    statement
```

The if-elif Statement →

Sometimes, we need to check another condition in case the test-condition of if evaluates to false

```
if < conditional expression > :
    Statement
elif < conditional expressions > :
    statement
```

and

```
if < conditional exp. > :
    Statement
elif < conditional exp > :
    Statement
else
    Statement
```

```
if runs >= 100 :
    print ("Batsman scored a century")
elif runs >= 50:
    print ("Batsman scored a fifty")
else :
    print ("Batsman has neither scored a century
           nor fifty")
```

* The range() Function :-

- range() is used with the loops of python.
- It generates a list which is a special sequence type
- The common use of range() is as follows:

```
range (<lower limits>, <upper limit> [,step value])
                # both limits should be integers
```

- range(0,5) will produce list as [0,1,2,3,4] default step-value will be +1.
- range (0,10,2) → [0,2,4,6,8]
- range (5,0,-1) → [5,4,3,2,1].
- range (5) → [0,1,2,3,4]

→ Membership operator → in and not in

∴ in operator is used with range() in for loops.

3 in [1,2,3,4] → will return True as 3 is contained.

5 not in [1,2,3,4] will return True as 5 is not in this

'a' in "trade" → True

```
line = input("Enter a line:")
string = input("Enter a string:")
if string in line:
    print(string, "is part of", line.)
else:
    print(string, "is not contained in", line.)
```

* Looping Statement →

for loop → for <variable> in <Sequence>:
    statements_to_repeat

```
for a in [1,4,7]:
    print(a)
    print(a*a)
```

Output :
1
1
4
16
7
49

The while Loop →

A while loop is a conditional loop that will repeat the instruction within itself as long as a Conditional remains True

Page No.
Date

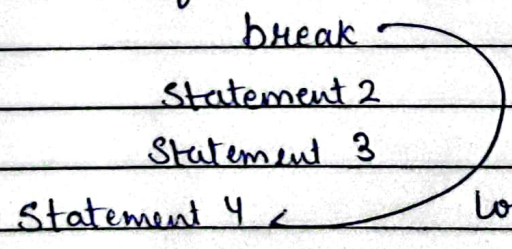
```
while <logical expression> :
    loop Body
a = 5
while a > 0 :
    print ("hello", a)
    a = a - 3
print ("Loop over!!")
```

The above code will print :

hello 5
hello 2
loop over!!

The break statement :-

```
for <var> in <sequence> :
    Statement 1
    if <condition> :
        break
    Statement 2
    Statement 3
Statement 4
Statement 5
```




```
# a = b = c = 0
for i in range (1, 21) :
    a = int (input (" Enter number 1 :"))
    b = int (input (" Enter number 2 :"))
    if b == 0 :
        print ("Division by zero error !")
        break
    else :
        c = a/b
        print ("Quotient =", c)
```

print ("Program over !")

The Continue statement :-

Continue Statement instead of forcing termination, forces the next iteration of the loop to take place, skipping any code in between.

```
a = b = c = 0
for i in range (0,3):
    print (" Enter 2 numbers")
    a = int ( input ("Number 1 :") )
    b = int ( input ("Number 2 :") )
    if b == 0 :
        print ("\n The denominator can not be
                zero. Enter again!")
        Continue
    else :
        c = a/b
        print ("Quotient =", c)
```

String :- individual characters of a string are accessible through the unique index of each character.

Traversing refers to iterating through the elements of a string, one character at a time.

Page No.
Date

```
name = "superb"
for ch in name :
    print ( ch, '-', end = '')
```

The above code will print :

s-u-p-e-r-b

Program to create (read) a string and display it in reverse order

```
# String1 = input ("Enter a string :")
print ("The", string 1, "in reverse order is :")
length = len(string 1)
for a in range (-1, (-length-1), -1) :
    print (string 1 [a])
```

Output → Enter a string : python
The python in reverse order is :

n
o
h
t
y
p

String operators :-

Basic operators : + and *

+ → Concatenation

* → replication

Page No.
Date

String Concatenation Operator +

"tea" + "pot" → teapot

'1' + '1' → '11'

"a" + 'o' → 'ao'

'123' + 'abc' → '123abc'

* Strings are immutable ; new strings can be created but existing strings cannot be modified.

* We cannot combine numbers and strings as operands with a + operator.

```
2+3 - 5          # addition - valid
'2'+ '3' = '23'  # Concatenation - valid
```

But the expression '2' + 3 is invalid

TypeError : cannot concatenate 'str' and 'int' Objects

String Replication Operator *

The * operator when used with numbers, it returns the product of the two no.

To use a * operator with strings, we need two types of operands - a string and a number, number * string or string * number.

3 * "go!" → 'go!go!go!'

"1" * 2 → "1"

Page No.
Date

Caution ! we cannot have strings as both the operands with a * operator.

```
2 * 3 = 6          # multiplication - valid
"2" * 3 = "222"    # replication - valid
```

But "2" * "3" is invalid

Typeerror : can't multiply sequence by non - int of type 'str'.

Membership operators :-

in → returns True if a character or a substring exists in the given string

not in → returns True if a character or a substring does not exist in the given string

"12" in "xyz" → False.
"a" in "heya" → ~~False~~ True
"j" in "Japan" → False

"jap" not in "Japan" → True
"123" not in "hello" → True
"123" not in "12345" → False

Comparison Operators :-

"a" == "a" → True
"abc" == "abc" → True
"a" != "abc" → True
"A" != "a" → True

Page No.
Date

Common Characters & their Ordinal values

'0' to '9' → 48 to 57
'A' to 'Z' → 65 to 90
'a' to 'z' → 97 to 122

'a' < 'A' → False
'ABC' > 'AB' → True
'abc' <= 'ABCD' → False

Determining Ordinal / Unicode value of a Single character

Python offers a built-in function ord() that takes a single character and returns the corresponding ordinal Unicode value.
It is used as per following format:

ord(<single-character>)

>>> ord('A') → 65

Opposite of 'ord' is 'chr'
while 'ord()' returns the ordinal value of a character.
chr() takes the ordinal value in integer and returns the character corresponding to that ordinal value.
The general syntax of chr() is

>>> chr(65) → 'A'

Page No.
Date

String Slices :→

The term 'string slice' refers to a part of the string, where strings are sliced using a range of indices.

	0	1	2	3	4	5	6
word	a	m	a	z	i	n	g
	-7	-6	-5	-4	-3	-2	-1

Then, word[0:7] → 'amazing'

(the letters starting from index 0 going up till 7-1 i.e. 6 ; from indices 0 to 6 both inclusive)

word[0:3] → 'ama'
word[2:5] → 'azi'
word[-7:-3] → 'amaz'

we can skip either of the begin-index or last.
. for missing begin-index

▲ Indexing works only from left to right.
Right to left is not possible.

s = "Hello"
print(s[5]) → error because 5 is invalid index - out of bounds, for string "Hello"

But if we give
s = "Hello"
print(s[4:8]) → One limit is outside the bounds
→ o

String functions and Methods →

1) len() → returns the no. of characters in string

A = "Hello India"

print (len(A)) → 11

2) Capitalize() → Returns a copy of the string with its first character capitalized

A = "hello india"

print(A. Capitalize())

↳ 'Hello india

3) String. isspace() → If all characters of the string are space it returns True otherwise returns false.

A = " "

print (A. isSpace()) → false

A = " "

print (A. isspace()) → True

A = "Ram Kumar" →

print (A. isspace()) → false

4) isalnum() → Returns True if the characters in the string are alphanumeric (alphabets or numbers) and there is at least one character, False otherwise.

>>> string = "abc123" → string. alnum() → True

>>> string2 = "hello" → string2. alnum() → True

>>> string3 = '12345' → string3. alnum() → True

>>> string4 = ' ' → string4. alnum() → false

Page No.
Date

5) Isalpha():- Returns True if all characters in the string are alphabets and there is at least one character, false otherwise.

```
String="abc123" >>> string.isalpha() → false
String2='hello' >>> String2.isalpha() → True
String3='12345' >>> String3.isalpha() → True
String4='' >>> String4.isalpha() → False
```

6) string.isdigit() → Returns True if all the characters in the string are digits. There must be at least one character, otherwise it returns false.

```
>>> string.isdigit() → false
>>> string2.isdigit() → False
>>> string3.isdigit() → True
>>> string4.isdigit() → False
```

7) string.islower() → Returns True if all characters in the string are lowercase. There must be at least one cased character. It Returns false otherwise.

```
>>> string='hello'
>>> string2='THERE'
>>> String3='Goldy'
>>> string.islower() → True
>>> string2.islower() → False
>>> string3.islower() → False.
```

8) string.isupper():- Tests whether all cased characters in the string are uppercase and requires that there be at least one cased

character. Returns True if so and False otherwise.

```
>>> string = "HELLO"
>>> string2 = "There"
>>> string3 = "goldy"
>>> string4 = "U123"
>>> string5 = "123f"

>>> string.isupper() → True
>>> string2.isupper() → False
>>> string3.isupper() → False
>>> string4.isupper() → True
>>> string5.isupper() → False
```

9) string.lower() :- Returns a copy of the string converted to lowercase.

```
>>> string.lower() → 'hello'
>>> string2.lower() → 'there'
>>> string3.lower() → 'goldy'
>>> string4.lower() → 'u123'
>>> string5.lower() → '123f'
```

10) string.upper() :- Returns a copy of the string converted to uppercase.

```
>>> string.upper() → 'HELLO'
>>> string2.upper() → 'THERE'
```

11) String.find(substring, [start], [end])

- Returns the lowest index in the string where the substring sub is found within the slice range of start and end.

- Returns -1 if substring is not found.

```
>>> String = 'it goes as - ringa ringa roses'
>>> sub = 'ringa'
>>> string.find(sub)
    13
>>> string.find(sub, 15, 22)
    -1
>>> string.find(sub, 15, 25)
    19
```

12) lstrip() :- It removes any leading characters (space is the leading characters to remove)

Syntax :- (optional)

string.lstrip(characters)

```
>>> string = "hello"
>>> string.lstrip() = 'hello'
>>> string2 = 'There'
    'There'
>>> string2.lstrip('the')
    'There'
>>> string2.lstrip('The')
    're'
>>> string2.lstrip('re')
    'There'
```

```
>>> string2.lstrip('Te')
'here'
>>> string2.lstrip('Teh')
're'
>>> string2.lstrip('heT')
're'
```

13) rstrip() :- The rstrip() works on identical principle as that of lstrip(), the Only difference is that it matches from right direction.

String = "hello"
String 2 = 'There'

```
>>> string
'hello'
>>> string.rstrip()
'hello'
>>> string2.rstrip('ere')
'Th'
>>> string2.rstrip('care')
'Th'
>>> string2.rstrip('car')
'There'
```

Page No.
Date

Modules :-

A module is a file containing pythm definition, functions, variables, classes and statements. The extension of the file is ".py".

Importing pythm modules →

(i) Using import statement

- import <module_name>
- import <module1>, <module2>, ... <module>
- To access a particular method from any module :

  ModuleName.functionName()

```
>>> import math
>>> math.sqrt(144) → 12.0
```

(ii) Using from statement : used to access specific / all attributes from a module.

- from <module> import <functionname>
- from <module> import <functionname1>, <fun2>... or
- from <module> import *
- To access a particular method from any module :

  functionName()

```
>>> from math import pow sin
>>> sin(0) → 0.0
>>> sqrt(23) → 'name' sqrt' is not defined
```

Page No.
Date

>>> from math import sqrt, sin, cos
>>> sqrt(90) = 0.8 9399

import math → math.sqrt()
from math import * → sqrt()

Module Aliasing :- (to give an alternate name)
- Renaming the module at the time of import, using as keyword.

- import <ModuleName> as <alias name>

# import math as m
>>> m.sqrt(255) → 15.98