Python
Basics to Advanced

Basics:

* Intro to python
* Print function
* Variables & Data Types
* Operators.
* Conditional statement
* Match case
* Loops

Intermediate:

* Strings
* List, Tuple
* Sets
* Dictionaries
* Functions
* Variable & method scope
* File & Exceptional handling

Advanced:

* Classes and objects
* Oops Concepts
* Functional programming
* Testing & scheduling
* Modules & packages
* Decorators, Re module
* Iterators & Generators

(Basics

1. Intro to python :

Definition :

* python is an interpreted language, high-level General purpose language.

↳ High level : easily understood by humans.

↳ Interpreted : To execute programs.

↳ General purpose : variety of applications.

History :

* Founder – Guido van Rossum
* Year – 1989
* purpose – Better Readability
* versions – 0.9, 2, 3.

Features :

1. Easy to write, read
2. free and open source
3. Interpreted
4. supports modularity
5. Extensible.
6. Dynamic type system
7. Automatic memory management
8. supports third party packages
9. Object oriented, Functional language.

modes :

↳ Interactive mode

↳ script mode

Applications :

1. Development - web, App
2. AI, ML, Data science
3. Automation, GUI.

2. print Functions:

* The print() function is used to display the Output in the Console.

* It supports custom separators, endings and flushing

* It can be intercepted, overriden or redirected.

Syntax:

Print(*objects, sep=' ', end=' ', file= , flush)

↳ *objects - One or more things to print
↳ sep - separator between Object
↳ end - what to append at the end
↳ file - where to write
↳ flush - Force immediate writing

Ex:

Print("Hello world!", sep=',', end='\n')

O/P:

Hello, world!

Uses:

* Debugging and logging

* writing logs directly to a file

* streaming Progress update

* Creating CLI-based tools

* Redirecting Outputs in unit tests

3. Variables:

* Variables are reference for the data or value.
* In python, a variable is not a box it's a label that points to an Object in memory.

Syntax:

name = value

Ex:

x = 10
↳ x - label pointing to the Object
↳ 10 - integer stored somewhere in memory.

Conditions:

* Can't starts with number
* Doesn't contain space instead use '_'.
* It is case sensitive.

Uses:

1. To store dynamic values
2. Reused computed results instead recalculate.
3. Pass values between functions / modules.
4. Reference mutable Objects.

Types:

↳ Global variable
↳ Protected variable
↳ Private variable
↳ Local and Non-local variable

4. Data Types :

* Every value in Py is an Object.
* It is defines what operations are allowed.
* Types can be dynamic but strongly typed.

Built in Data Types :

a) Numeric Types

↳ int	-	Arbitrary precision integers (42)
↳ float	-	64 - bits floating point. (1.5)
↳ complex	-	complex numbers. (2+3j)

b) sequence Types

↳ str	-	Immutable text
↳ list	-	mutable, ordered collection
↳ tuple	-	Immutable, ordered collection.
↳ range	-	Efficient sequence Generator.

c) set & map types

↳ set	-	unordered, unique elements
↳ frozenset	-	Immutable set
↳ dict	-	key- value pairs

d) other types

↳ bool	-	True or False
↳ None	-	No value
↳ bytearray	-	arrays in bytes
↳ byte	-	Contains byte values.

5. Type casting :

* Type conversion is the process of converting one data type to another.

* Two types of type conversion :-

a) Implicit conversion.

b) Explicit conversion.

Use case :

1. Working with user input.
2. Interfacing with databases / APIs where types differ.
3. Avoid TypeErrors
4. Converting for performance.

Ex :

```
print(int("10"))          # 10
print(float("3.14"))      # 3.14
print(str(42))            # "42"
print(list("abc"))        # ['a','b','c']
```

a) Implicit Conversion :

* Py automatically converts one data type to another without any user involvement.

* It is also called type promotion.

b) Explicit conversion :

* The data type is manually changed by the user as per requirements.

* Conversion with built in functions.

6. Input() Functions

* User inputs typically collected using input()

* input() always returns a string. need Casting

Various sources:

↳ Keyboard
↳ Command line arguments.
↳ files, sockets, stdin requirements.
↳ GUI or apps.

*args and ** kwargs:

↳ *args: accepts any number of positional arg as tuples
↳ ** kwargs: " as dictionary

Ex:

```
def demo(*args, ** kwargs):
    print(args)
    print(kwargs)
demo(1, 2, 3, name = "Hi", age=20)
```

O/P:

```
(1, 2, 3) {'name': 'Hi', 'age': 20}
```

Syntax:

```
x = input()
```

7. Operators:

* Operators are symbols or keywords that perform operation on values

* Operands are values on which the operator is applied (a+b).

Types Of operators:

a) Arithmetic Operators:

+	-	Addition	-	$x+y$
-	-	Subtraction	-	$x-y$
*	-	multiplication	-	x*y
/	-	Division	-	x/y
//	-	floor division	-	$x//y$
**	-	power	-	x**y
%	-	modulus	-	$x\%y$.

b) Comparison operator:

>	-	Greater than	-	$a>b$
<	-	Lesser than	-	$a<b$
>=	-	Greater/equal	-	$a>=b$
<=	-	Lesser/equal	-	$a<=b$
==	-	equal to	-	$a==b$
!=	-	Not equal	-	$a!=b$

c) Logical Operators:

and -	True if both are True	- x and y
or -	True if either one is true	- x or y
not -	True if it is false	- not x

d) Bitwise Operators:

& -	Bitwise AND
| -	Bitwise OR
~ -	Bitwise NOT
^ -	Bitwise XOR
>> -	Bitwise right shift
<< -	Bitwise left shift

e) Assignment Operators:

+= -	simple design
= -	add and assign
-= -	subtract and assign
*= -	multiply and assign
/= -	divide and assign
//= -	floor divide and assign
%= -	modulus and assign
**= -	power and assign
&!= -	Bitwise Variants

8. Conditional statements:

* Conditions to make decisions whethere to execute or nets on the block.

* The decision is made based on the condition given which returns True or False.

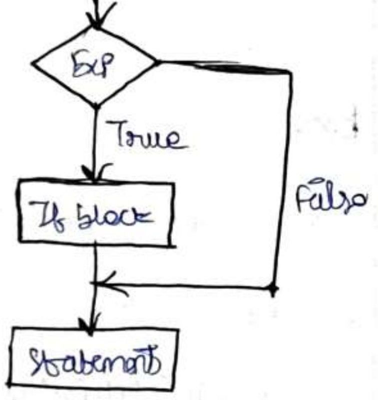
a) IF statements:

* This only executed the given condition is True.
* If it is False, it's skipped.

Syntax:

```
if condition:
    # this block executed.
    # If condition is True.
```

Flowcharts:



Ex:

```
a = int(input("Enter a Number: "))
if a>5:
    print("your entered number is greater than 5")
print("Thank you")
```

O/p:

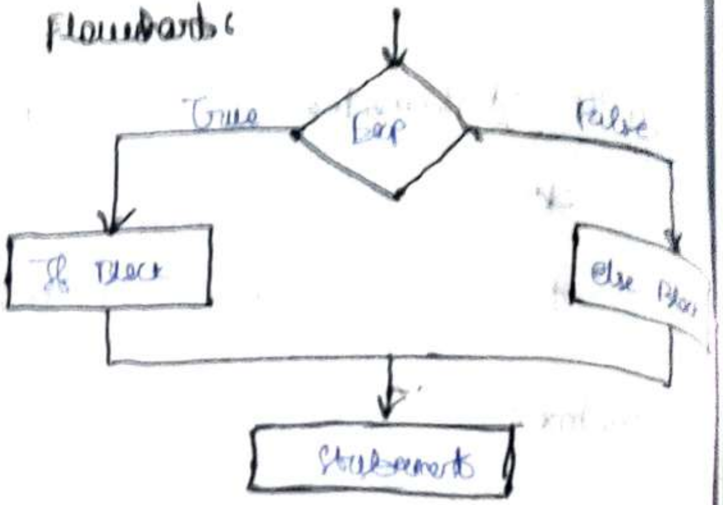
```
Enter a number :10
you entered a number is greater than 5
Thank you!
```

b) If - Else statements :

* The If block executes only condition is True otherwise it executes to else as False condition.

* else is alternate condition and doesn't take any conditions.

Flowchart:



Syntax:

```
if condition:
    statement 1
else:
    statement 2
```

Ex:

```
age = int(input("Enter your age: "))
if (age >= 18):
    print("Elligible for vote")
else:
    print("Inelligible for vote")
print("Thank you!")
```

O/P:

```
Enter your age: 17
Inelligible for vote

Enter your age: 20
Elligible for vote
Thank you!
```

c) If - Elif - Else statements:

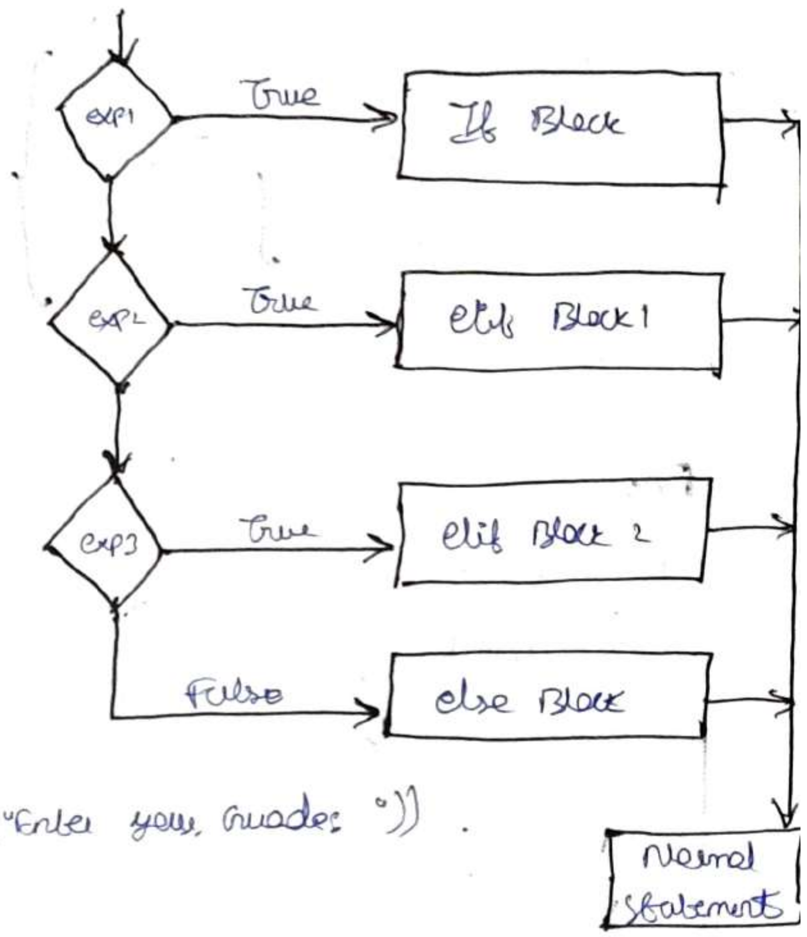
* If there are two or more alternates the elif block executed and if - elif - else statement is used.

* If both the If and elif statement is not True only execute the else statement.

Syntax:

```
if Condition:
    # if block
elif Condition:
    # elif block
elif Condition:
    # another elif
Else:
    # else block
```

Flowcharts:



Ex:

```
grade = float(input("Enter your Grade: "))
if (grade > 80):
    print ("Grade A")
elif (grade > 60):
    print ("Grade B")
elif (grade > 40):
    print ("Grade C")
else:
    print ("Fail!")
```

O/P:

```
Enter your Grade: 80
Grade B
```

d) Nested If Statements:

* If statement contain another If statement.

* Only used in situation where there are multi
Condition to check.

Syntax:

```
if Condition:
    statement1
    if Condition2:   }  Inner If  }  Outer If
        statement2   }  Block     }  Block
    ......
    ......
......
```

Ex:

```
ac = input("Enter Ac/non-Ac").lower()
food = input("Enter veg/non-veg").lower()
if (ac == "ac"):
    if(food == "food"):
        print("ac veg")
    else:
        print("ac non-veg")
else:
    if (food == "veg")
        print("non-ac veg")
    else:
        print("non-Ac non-veg")
```

O/P:

```
Enter Ac/non-Ac: non-Ac     }  non-Ac non-veg
Enter veg/non-veg: non-veg  }
```

e) matlah case :

* The match case is an alternative to the if-elif-else statements.

* It compare a give variables value to each case until the pattern matches.

Syntax :

```
variable name = " "
match variable_name :
      case <pattern1> :
            statement1
      case <pattern 2> :
            statement2
      case <pattern n> :
            statement3
```

Ex :

```
num = int(input("Enter 1,2, or 3 "))
match num :
    case 1 :
        print(" Success... ")
    case 2 :
        print (" Warning... ")
    case 3 :
        print (" Error... ")
    case _ :
        print ("please enter 1,2 or 3")
```

O/P :

```
Enter 1, 2, or 3 : 1
Success...
```