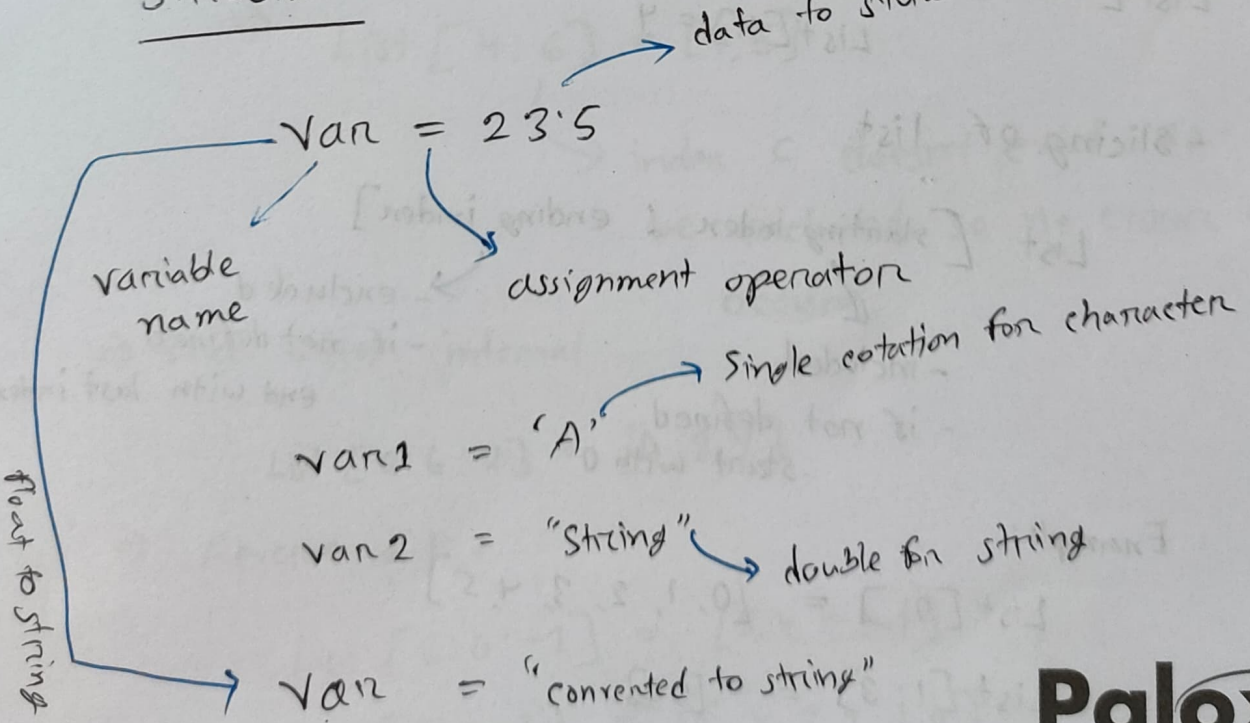
Python

Variable:

- It's like a container to store or hold some data likes number, text, string, character etc.
- In python, we don't need to define any datatype before variable assignment.
  - Python automatically detect them.
- variable name can't start with a digit
  - also you can't use a reserve word as a variable name.

Structure:



✱ Basic Data Structure:

- The way, ~~we~~ how we store data.

Some data structure:

- List ⇒ most popular
- Graph
- Tree
- Linked List ⇒ its also popular
- Tuples

⇒ List ⇒ list of data or array in C.

List = [0, 1, 2, 3, 4, 5]

name of list ← List = [0, 1, 2, 3, 4, 5] → data

⇒ List access:

List [index] ⇒ List [0] = 0
List [4] = 4

- Slicing of list

List [starting index : ending index]

- starting index → included
  - if not defined start with 0
- ending index → excluded
  - if not defined end with last index

Example:

List [0:] = [0, 1, 2, 3, 4, 5]

List [1:3] = [1, 2]

List [2:2] = [] // empty list

⇒ Negative index!

Lets ~~index~~ length = 6

List[-4] = List[2] = 2

↳ ~~index~~ length - 4 = 2

List[-3] = length - 3 = 3

List[-0] = List[0]

List[-7] = -1 ⇒ out of index

Combination of slice with negative index

List[1:-5] = []

List[1:-4] = [1]

⇒ List[4:6] = [4,5]

↳ index 6 doesn't exist, as it will be excluded, so no error will occure.

⇒ Slice with interval

List[0:6:2] = [0,2,4]

⇒ Reverse a list

List[6:0:-1] = [5,4,3,2,1]

List[6::-1] = [5,4,3,2,1,0]

⇒ extend a list

List.append(6)

List = List + [6]

List = List + [6,7]

List = List + List2

⇒ adding new item in a specific index

⇒ List[:4] = [10]

↳ as 4 is excluded.
new item will insert in
index 3 and the data ~~from index 3 will shift to right.~~
of previous index will be
remove.

List[4:4] = [10]

⇒ in this case new data will
insert in index 4 and data from
index 4 will shift to right.

⇒ Actually it is replacing the data of given
range with your provided data.

List [2:4] = [10]

⇒ starting with index 2 and end with index 3, all data will be remove and your data will place in this position.

⇒ Some function or method.

append(content) ⇒ adds new element at the end (content → element)

clear() ⇒ remove all elements from the list

copy() ⇒ returns a copy of the list

count(elements/content) ⇒ returns number of elements with the given content or value

extend(elements) ⇒ add the element at the end

index(content/value/elements) ⇒ return index where the content first appear

insert(index, content) ⇒ add the content at the given index, others will shift to the right.

pop(index) ⇒ remove the elements of given index

remove(content) ⇒ remove the element, where it appears first.

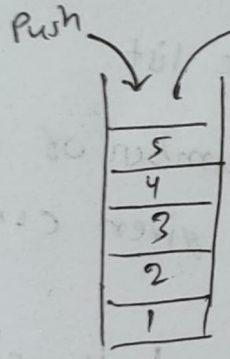
reverse() ⇒ reverse the list

sort() ⇒ sorts the list

Stack & Queue:

Stack

- LIFO ⇒ Last in First Out
- PUSH ⇒ adds an item in the top of the stack
- POP ⇒ return and ~~out~~ remove the top item.

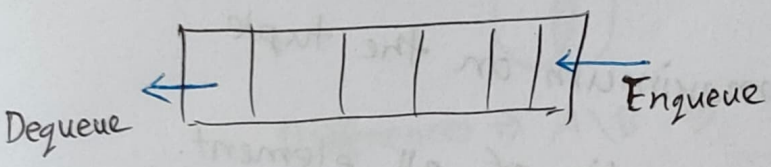



```
stack = []
stack.append(content) ⇒ PUSH
stack.pop() ⇒ POP
```

↳ no index given.
- then it will remove and return content of the last index

Queue

- First in, first out
- Enqueue ⇒ add an item at the end of the list
- Dequeue ⇒ remove an item from the starting index of the list




```
queue = []
queue.append(content)
        ↳ Enqueue
queue.pop(0) → Dequeue
        ↳ remove and return content of index 0.
```

Tuple:

~~Diff~~

- Same as list, a little difference in structure.

Tuple = (0, 1, 2, 3, 4, 5) → use parenthesis

- Use same slice operation as list.

Some method:

~~count() ⇒ rel~~

count (element) ⇒ count the element on the tuple

index (element) ⇒ find the index of the given element.

len ( ) ⇒ return the length of the tuple

min ( ) ⇒ return the minimum on the tuple

max( ) ⇒ return the maximum on the tuple

sum( ) ⇒ return the sumation of all element

sort( ) ⇒ sort the tuple

loop tuple ⇒ loop all element on tuple

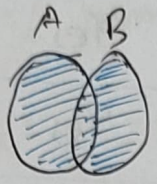
Set

set = { 20, 'Jessa', 35.75}

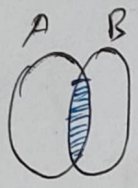
- contain all types of data, and all combination
- set doesn't allows duplicates items.

Set operation

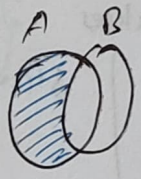
Union ⇒ $A \cup B$ ⇒ set of all values that are member of A or B or both



Intersection ⇒ $A \cap B$ ⇒ set of all values that are member of both A and B



Differences ⇒ $A \setminus B$ ⇒ Set of all values that are not members of B



Symmetric Difference ⇒ $A \Delta B$ ⇒ set of all values that are in one of the sets but not both.

Structure:

set. union (set2)
set. intersection (set2)

Difference between List, Tuple, Set, Dictionary

	Ordered	Mutability	Syntax	Slicing	Duplicate Element
List	Yes	Yes	[ ]	Yes	Yes
Tuple	Yes	No	( )	Yes	Yes
Set	No	No	{ }	No	No
Dictionary	No	Yes	{ }	No	No

Dictionary:

- Pair of data → key, value
- unordered
- optimized to retrieve values

key	Value
1	Python
2	Java
3	C++

Structure:

Dict = { key: value, key: value }

Example:

```
Dict = {
        'Name' : 'Joy',
        'Location' : 'Dhaka',
        'ID' : 2211424
       }
```

Dictionary access:

Dict["key"]

⇒ Dict['Name'] = "Joy"

⇒ Dict['ID'] = 2211424

⇒ Nested Dictionary:

```
Dict2 = {
        'Program' : "BS CSE",
        'Details' : Dict
        }
```

Access:

Dict2["Details"]["Name"] = Joy

Some methods:

clear()	keys()
copy()	update()
get()	pop()
items()	values()
fromkeys()	popitem()

Math Operators:

Addition ⇒ x+y

Subtraction ⇒ x-y

Multiplication ⇒ x*y

Division (float) ⇒ x/y

Division (floor) ⇒ x//y

Modulus ⇒ x%y
- returns remainder

Power ⇒ x**y = x^y

Logical Operator:

which will execute first ↑

NOT = not x

AND = x and y

OR = x or y

Comparison Operator

Equal ⇒ x==y

Not equal ⇒ x!=y

Greater than ⇒ x>y

Less than ⇒ x<y

Greater than or equal ⇒ x>=y

Less than or equal ⇒ x<=y

Control Flow:

if - else

```
if (condition):
    ....
    ....
```

indentation must

```
if (condition):        # can be combination of and or
    ....
    ....
else:
    ....
    ....
```

Nested if-else:

```
if  i<j:
    if j<k:
        i = j
    else:
        j= k
else:
    if j>k:
        J = i
    else:
        l=k
print (i,j,k)
```

i = 3
j = 5
k = 7

} 5 5 7

Single line if-else:

print ("x") if x >> else print ("Y")

print ("x") — body of if
if x >> — condition
else print ("Y") — body of else

condition
↓
← True
False →

⊛ Loops:

two types:

(i) For → if we know iteration count
(ii) While → if we know the condition only

Syntax:

List, tuple etc

for variable_name in iterable:

- intended
- each time it will take one element from iterable and assign it in the variable_name. When the given list not iterable, it will not execute

else:

if loop is not break, executed successfully and terminate successfully then it will execute.

function return number from 0 to 10

for i in range(10)

range (2, 5) ⇒ returns a list [2, 3, 4]

excluded

included

while condition:

else:

while n: → single variable, when value variable is not 0, it is true. only at 0, it is false

* range(2, 0, -1) ⇒ [2, 1]
  - Ending
  - difference
  - Starting

*

sum += i > sum

it will execute first and return 1 or 0
(1 → True, 0 → False)

sum += 1/0 ⇒ sum = sum + 1/0

* For breaking out from a loop ⇒ break

  for continue loop without executing bellow line ⇒ pass

* Functions:

  - some function may return value, some may not. depends how we define the function.

Syntax: (define)

```
def name(parameter):
    ....
    ....
    ....
    ....
```

```
def name(Parameter)
    ....
    ....
    ....
    return variable/value
```

Lamda function:

keyword

Parameter

```
func = lamda x,y : x+y
```

Statement / Expression → Single

Object Oriented programing:

Class → Blue Print of a object

Object_name → just a name lik variable name, we can give a object name known as instance.

Properties → details about the object. its a set of variable or data type.

Methods → functionality can be done by the object.

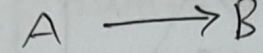
Concept of OOP

Inheritance

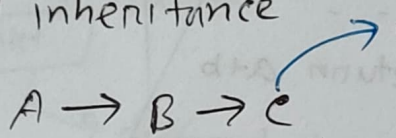
inherit properties of parent class.

5 types of inheritance available in python

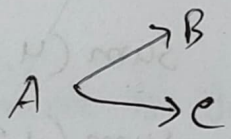
(i) Single inheritance



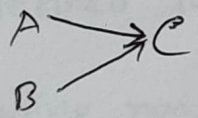
(ii) Multilevel inheritance



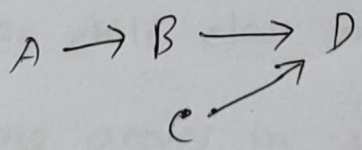
(iii) Hierarchical inheritance



(iv) Multiple inheritance



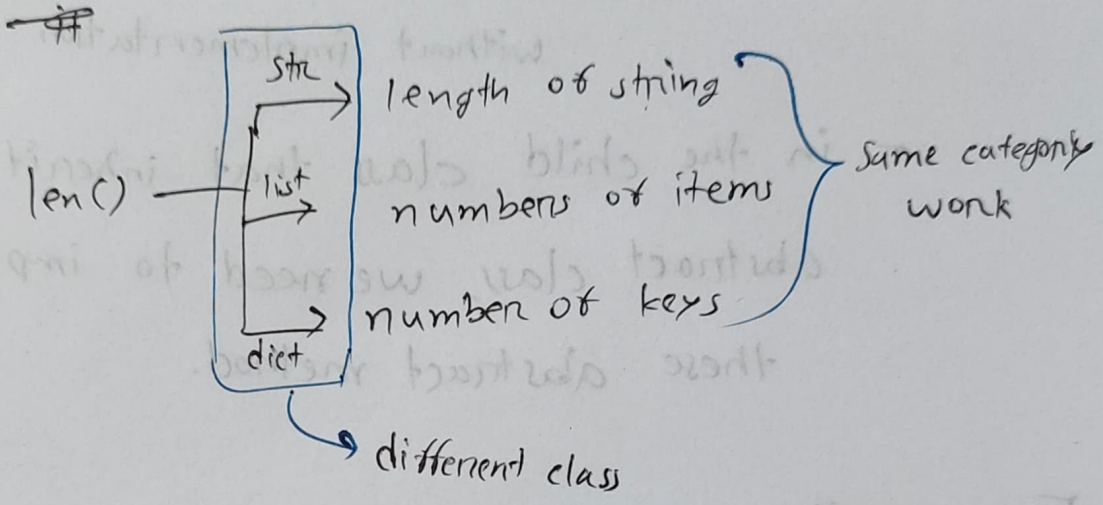
(v) Hybrid inheritance



Polymorphism:

- function name is same → class
- But the data type on object is different

⇒ Different class using the same function name.



Overloading:

- function name is same
- parameters different

⇒ last defined function will work in python

- overloading normally implemented in child class, to overwride the parent function.

  As child class may have more properties.

⇒ Overloading occurs in single class (in Python not possible)

  Overriding occurs in child class.

Data abstraction:

- hides the complex implementation.
- its a template for other class
  - provides list of methods and parameter without implementation
- in the child class that inherit the abstract class, we need to implement these abstract method.

Encapsulation:

- self-sufficient function - can work independently
- well-define readable code
- prevents accidental modification on deletion as well as security

⇒ use access specifier

	own class	derived class	object
Private	✓	×	×
Protected	✓	✓	×
Public	✓	✓	✓