syntax to get the list of keywords

In [5]:
```
import keyword
python_keywords = keyword.kwlist
print(python_keywords)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'globa
l', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
'return', 'try', 'while', 'with', 'yield']
```

python variables

python variable is a container that stores the assigned data. Once a data is assigned to a variable, it can be referred to by that name.

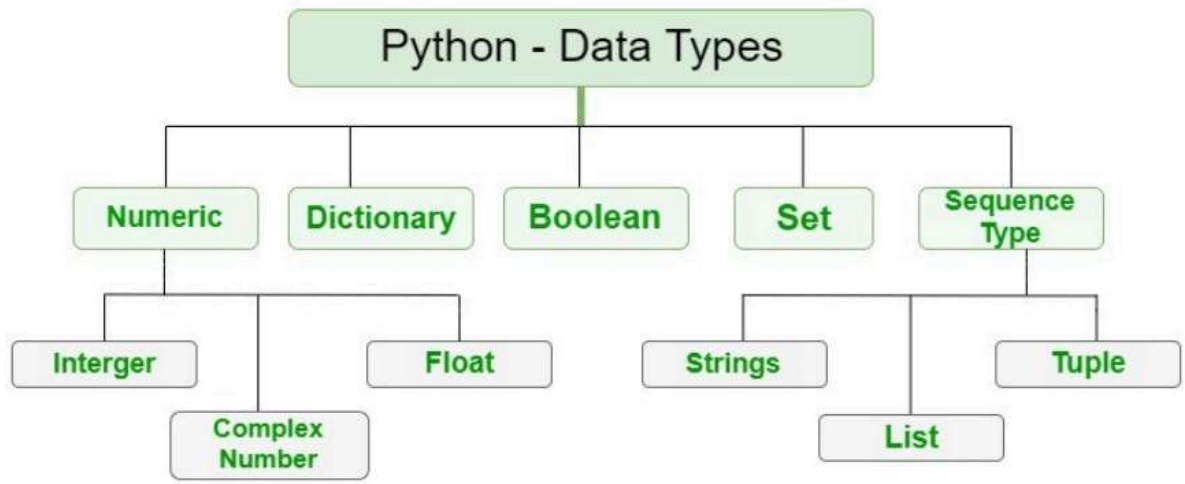
rules to write variables

-> A python variable contains alphanumeric and _ -> A python variable should not start with numbers -> Python variables are case senstive -> keywords cant be used as variables

In [8]:
```
abc = 10
hi_1 = 'Hi how are you?'
print(abc)
print(hi_1)
```

```
10
Hi how are you?
```

python data types



Name	Type	Description
Integers	int	Whole numbers, such as: 3 300 200
Floating point	float	Numbers with a decimal point: 2.3 4.6 100.0
Strings	str	Ordered sequence of characters: "hello" 'Sammy' "2000" "楽しい"
Lists	list	Ordered sequence of objects: [10,"hello",200.3]
Dictionaries	dict	Unordered Key:Value pairs: {"mykey" : "value" , "name" : "Frankie"}
Tuples	tup	Ordered immutable sequence of objects: (10,"hello",200.3)
Sets	set	Unordered collection of unique objects: {"a","b"}
Booleans	bool	Logical value indicating **True** or **False**

We can know the data type of the variable using type() function. We can also convert the int and float data types to complex, but not vice versa.

In [9]:
```
x = 2024
y = 24.2
z = 2j+24

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

In [11]:
```
print(complex(x))
print(complex(y))
print(int(z))
print(float(z))
```

```
(2024+0j)
(24.2+0j)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[11], line 3
      1 print(complex(x))
      2 print(complex(y))
----> 3 print(int(z))
      4 print(float(z))

TypeError: int() argument must be a string, a bytes-like object or a real number, not 'complex'
```

In [4]:
```
# python booleans

print(bool("hello"))
print(15<9)
```

```
True
False
```

python operators

There are totally 7 types of python operators. They are used to perform operations on variables and values.

Arithematic, assignment, comparision, logical, identity, membership, bitwise operators.

python arithematic operators

It does all the arithmatic operations between the variables

In [6]:
```
a = 2
b = 10
print(a+b)
print(a-b)
print(a*b)
print(a/b) #return quotient
print(a%b) # return remainder (modulus)
print(a**b) #exponent
print(a//b) # floor division - int from quotient
```

```
12
-8
20
0.2
2
1024
0
```

python assignment operators

In [7]:
```
a = 2 #assigning value 2 to the variable a
a = a+2 #assign sum of a+2 to a
print(a)
```

```
4
```

similarly, we have subtraction assignment(-=), multiply(*=), divide(/=), remainder(%=) and exponent (**=) assignment operator

python comparision operators

They compare two variables or values, and give the output in boolean form

In [12]:
```
a = 10
b = 5
print(a==b)
print(a!=b)
print('a>=b =', a>=b)
print('a<=b =', a<=b)
print('a>b =', a>b)
print('a<b =', a<b)
```

```
False
True
a>=b = True
a<=b = False
a>b = True
a<b = False
```

python logical operators

They are used to check if the statement is true or false. They will combine two or more conditions and give the output.

"and" operator gives if both the conditions are true. "or" operator gives true if atleast any one condition is true. "not" operator gives true if the condition is false and viceversa.

In [15]:
```
a = 10
print(a>2 and a<15)
print(a>2 or a>15)
print(not(a>2 and a<15))
```

```
True
True
False
```

python identity operators

"is" op gives true if both operands are identical. "is not" op gives true if both operandss are not identical.

In [16]:
```
a = 10
b = 9
c=10
d="vaishnavi"
e ="Vaishnavi"
print(a is not b)
print(a is c)
print(b is c)
print(d is not e)
```

```
True
True
False
True
```

python membership operators

They are used to check if the value or variab;e is present in the sequence or not

In [23]:
```
A = "Hello world!"
B = {1:'a', 2:'b'}
print('a' in B )
print('Hello' not in A)
print( "!" in A)
```

```
False
False
True
```

python bitwise operators

In Python, bitwise operators are used to perform bitwise calculations on integers. The integers are first converted into binary and then operations are performed on each bit or corresponding pair of bits, hence the name bitwise operators. The result is then returned in decimal format. Python bitwise operators work only on integers.

OPERATOR	NAME	DESCRIPTION	SYNTAX
&	Bitwise AND	Result bit 1,if both operand bits are 1;otherwise results bit 0.	x & y
|	Bitwise OR	Result bit 1,if any of the operand bit is 1; otherwise results bit 0.	x | y
~	Bitwise NOT	inverts individual bits	~x
^	Bitwise XOR	Results bit 1,if any of the operand bit is 1 but not both, otherwise results bit 0.	x ^ y
>>	Bitwise right shift	The left operand's value is moved toward right by the number of bits specified by the right operand.	x>>
<<	Bitwise left shift	The left operand's value is moved toward left by the number of bits specified by the right operand.	x<<

precedence

Operators have different levels of precedence, which determine the order in which they are evaluated. When multiple operators are present in an expression, the ones with higher precedence are evaluated first. In the case of operators with the same precedence, their associativity comes into play, determining the order of evaluation.

Operator	Description	
( )	Parentheses (grouping)	Highest
**	Exponentiation	
~x	Bitwise nor	
+x,-x	Positive , negative (unary +,-)	
*,/,//,%	Multiplication , division , floor division , remainder	
+,-	Addition , subtraction	
&	Bitwise and	
^	Bitwise XOR	
|	Bitwise OR	
<,<=,>,>=,<>,!=,== ,is,isnot	Comparisons (Relational operators),identity operators	
not x	Boolean NOT	
And	Boolean AND	
or	Boolean OR	Lowest

Python data types

Data type	Ordered	Mutable	constructor	Example
List	yes	yes	[ ]	[1, 2, 'yes' , 0.1]
String	yes	No	" " or ' '	a = 'Vaishnavi' ; b = "how are you? "
Tuple	yes	No	( )	(1, 2, 'yes' , 0.1)
Set	No	Yes	{ }	{1,2,3,4}
Dictionary	No	yes	{ key : value}	item = { 'name' : ' Vaishnavi'}

conditional statements

conditional statements are those, used to control the flow of a program using certain conditions. There are three types of conditional statements. if, else, if else (elif).

if condition

if the condition is satisfied, it gives the required output. if the condition is not satisfied, it simply skips the execution.

In [ ]:
```
#syntax

if condition:

    body of statement
```

In [24]:
```
colour = "Blue"

if colour == "Blue" :
    print('True')
```

```
True
```

else condition

If the 'if' condition is satisfied, it gives the respective output. If it is not satisfies, it gives the output corresponding to else statement

In [ ]:
```
#syntax

if condition:

    body of statment

else:

    body of statement
```

In [26]:
```
#else condition

colour = "Blue"

if colour == "blue" :
    print('True')
else :
    print('False')
```

```
False
```

if else (elif) condition

If the 'if' condition is not satisfied, then it executes if else condition. if this is satisfied, it gives the respected output. or else it gives the oouput corresponding to else condion.

In [ ]:
```
#syntax

if condition:
    body of the statement
elif:
    body of the statement
else:
    body of the statement
```

In [27]:
```
#ELIF condition

colour = 'Blue'
if colour == 'blue':
    print ('yes')
elif colour == 'BLUE':
    print ('no')
else :
    print ('undefined')
```

```
undefined
```

nested if statements

if-else pair of conditon is used here. it executes one-by-one pair of conditions.

In [28]:
```
age = int(input("Enter your age:"))

if age >= 18 :
    if age > 18 :
        print ('eligible')
    else :
            print ('waiting list')
else :
    print ('not eligible')
```

```
Enter your age:25
eligible
```

While loop

This loop is used to run a certain block of code, until certain condition is met.

In [ ]:
```
#syntax

while condition:
    body of statement
```

In [29]:
```
#While loop

i = 4
while i<= 8 :
    print(i)
    i = i + 2
```

```
4
6
8
```

For loop

This loop is used to iterate a sequence. That can be done on either list, tuples, dictionary, string or range.

In [ ]:
```
#syntax

for value statement:
    body of statement
```

In [30]:
```
for x in range (2,5):
    print (x)
```

```
2
3
4
```

control statements

They provide the ways to control the loop or to handle certaiin situations.

pass

This statement is used when, no execution is requied at that particular step. It operates on the loop behind, but the output at that certain point is not given.

In [31]:
```
#pass

for i in range(5):
    if i ==1 :
        pass
    else:
        print(i)
```

```
0
2
3
4
```

break

This statement is used, when we need to terminate the loop immediately. Then the operation goes to the next existing loop.

In [32]:
```
for i in range (10):
    if i == 2:
        break
    print(i)
```

```
0
1
```

continue

This is similar to pass statement. This skips current iteration and moves to next iteration.

In [33]:
```
for i in range (10):
    if i ==3:
        continue
    elif i ==8:
        continue

    else :
        print (i)
```

```
0
1
2
4
5
6
7
9
```

python functions

A function is a block of code that performs a specific task. Data can be passed into functiona as a parameter. It returns data as a result. There are 2 types of functions. In-built and user defined functions. In-built functions are already available in python which are ready to use. User defined funactions are created based on our requirement.

In [ ]:
```
#Syntax

def fn_name(argument):
   #function body
   return
```

In [34]:
```
#simple function

def greet(name):
    print(f"Happy Birthday {name}!")

#calling function
greet("Vaishu")
```

```
Happy Birthday Vaishu!
```

python lambda function

A lambda function can take n number of arguments, but have only one expression.

In [ ]:
```
#syntax

lambda arguments : expression
```

In [35]:
```
x = lambda a : a+20
print(x(10))
```

```
30
```

Global and local variables

Python local variables are defined inside a function and are applicable only to that particular block of function. Where are, global variables are defined not to any particular block of function, but are applicable globally through out the program.

In [36]:
```
#local variable in python

def f():
    s = "I am Vaishnavi"
    print(s)
f()
```

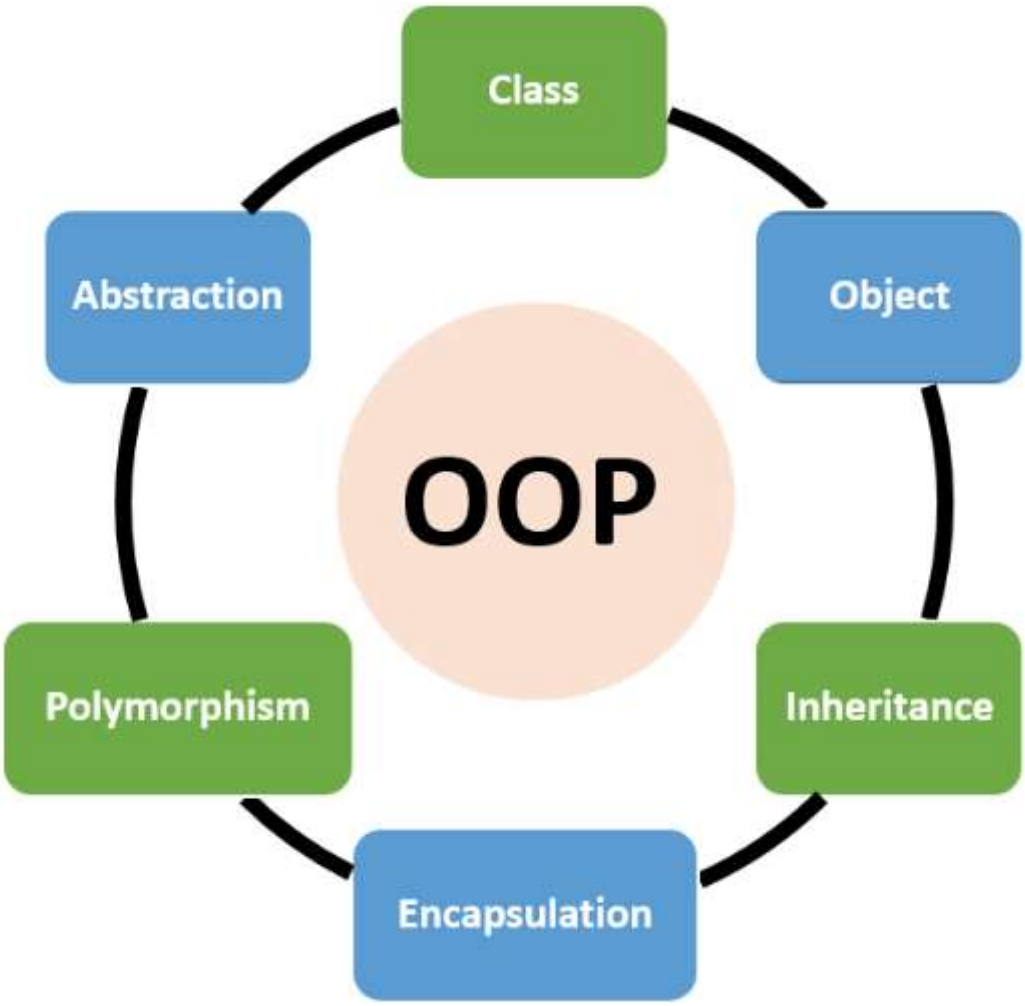
```
I am Vaishnavi
```

Python global variable

In [37]:
```
def f():
    print("local Vble",s)
s = "I am Vaishnavi"
f()
print("Global Vble",s)
```

```
local Vble I am Vaishnavi
Global Vble I am Vaishnavi
```

python object orientec programming language



In Python, object-oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

class

Class is like a constructor. or a blueprint to for creating objects

In [41]:
```
class movie:
    def __init__(self, name, date):
        self.name = name
        self.date = date
    def hit(self):
        print("Hit")
movie1 = movie("Hi nanna", "friday")
movie2 = movie("Hanuman", "friday")
```

```
print(f"{movie1.name} is released on {movie1.date}")
movie2.hit()
```

```
Hi nanna is released on friday
Hit
```

The self parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class. It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class.

The Init() function

Basically this is the constructor or which initializes the attributes of the object

Inheritance

A new class will be derived from already existing class. Derived class is called child class and the class from which it is derived is called parent class. Child class has all the properties of parent class and we can add extra properties also to the child class. This avoids to write the code again and again. So the parent class has only its properties., whereas, the child class owns both the parent class properties and its own properties.

Encapsulation

This is a methos where methods and variables are buldled together into a single unit. This helps in protecting or hiding the data from other classes. It also avoids mistake edits or changes in the code and unneccesary access from other classes.

Polymorphism

This is the method where a single entity (method/object/variable) can be used in multiple variable forms. For example, take polygon. It can be in the form of square, pentagon, hexagon, septagon, octagon etc.

Abstraction

This is the method where execution of code is hidden. We can also say it as all the code running behind will be hidden only output is focused here.