دانشگاه شهید بهشتی ۱۳۳۸

Advanced Programming

winter 2024

Maryam Babaei

WHAT IS PYTHON?

WHAT IS PYTHON?

- An **interpreted**, **high-level**, **general-purpose** programming language
- Dynamic type
- Garbage collection
  - reference counting
- It supports object-oriented and functional programming

INTRODUCTION TO PYTHON

DATA TYPES

- Text Type
  - str
- Numeric Types
  - int, float, …
- Boolean Type
  - bool
- Void Type
  - None

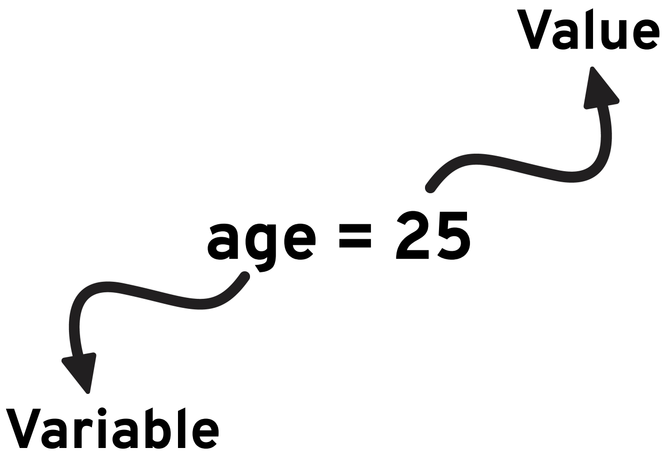
```python
a = "Hello"           # str
b = 10                # int
c = 3.14              # float
d = 2 + 3j            # complex
f = True              # bool
g = None              # NoneType
```

- Comments
  - # symbol for one-line comments
  - """ or ''' For multi-line comment

TYPE CASTING

Function	Conversion
int()	string, float ->int
float()	string, int -> float
str()	int, float - >string
complex()	int, float -> complex

VARIABELS



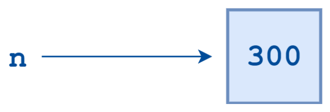
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- Variable names are case-sensitive (age, Age and AGE are three different variables)

RESERVED WORDS

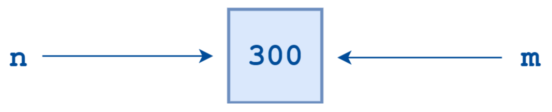
False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

OBJECT REFERENCES

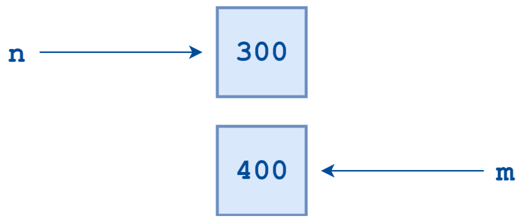
```
>>> n = 300
```



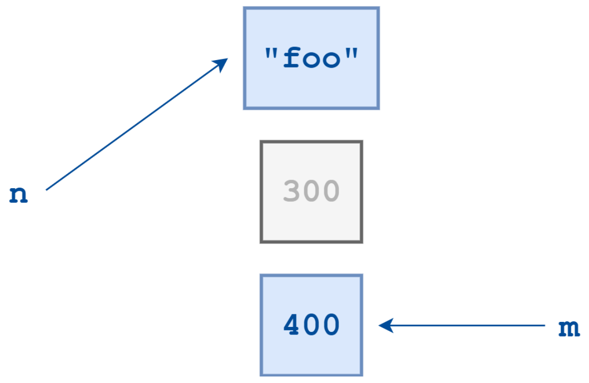
```
>>> m = n
```



```
>>> m = 400
```



```
>>> n = "foo"
```



OBJECT IDENTITY

```
>>> n = 300
>>> m = n
>>> id(n)
60127840
>>> id(m)
60127840

>>> m = 400
>>> id(m)
60127872
```

```
>>> m = 300
>>> n = 300
>>> id(m)
60062304
>>> id(n)
60062896

>>> p = 30
>>> q = 30
>>> id(p)
1405569120
>>> id(q)
1405569120
```

BASIC OUTPUT

```python
age = 25
print(age)   #output is 25

age, height = 25, 170
print(height)  #output is 170

age = height = 25
print(height)  #output is 25
```

```python
age = 25
print(type(age))   #output is <class 'int'>

height = 170.5
print(type(height))   #output is <class 'float'>

name = "ali"
print(type(name))  #output is <class 'str'>

alive = True
print(type(alive))  #output is <class 'bool'>
```

BASIC INPUT

```
>>> user_input = input()
foo bar baz
>>> user_input
'foo bar baz'
```

```
>>> number = input("Enter a number: ")
Enter a number: 50
>>> print(number + 100)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int

>>> number = int(input("Enter a number: "))
Enter a number: 50
>>> print(number + 100)
150
```

BASIC INPUT

```python
a = int(input("Enter a number: "))  #Enter a number: 1
b = float(input("Enter a number: "))  #Enter a number: 1
c = str(input("Enter a number: "))  #Enter a number: 1

print(a, type(a)) #output is 1 <class 'int'>
print(b, type(b)) #output is 1.0 <class 'float'>
print(c, type(c)) #output is 1 <class 'str'>
```

BASIC OPERATIONS

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

ASSIGNMENT OPERATION

Operator	Example	Same As
=	x = 5	x = 5
:=	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 **=2	x = x ** 2

COMPARISON OPERATIONS

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

LOGICAL OPERATIONS

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

SIMPLE EXAMPLE

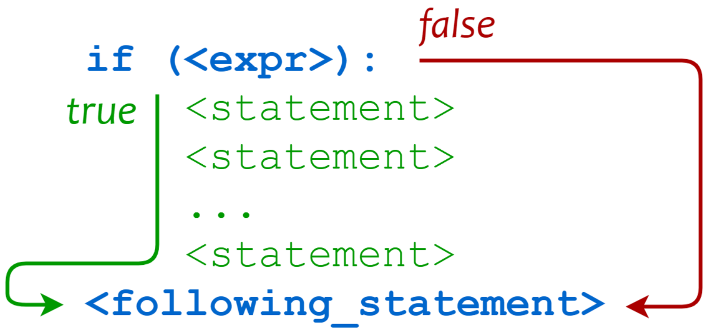
```python
#Calculate your BMI
weight = float(input("Enter your weight in kilograms: "))
#Enter your weight in kilograms: 52
height = float(input("Enter your height in meter: "))
#Enter your height in meter: 1.7

BMI = weight/(height ** 2)
print("yout BMI is: ", BMI) #yout BMI is:  17.993079584775087
```

CONTROL STRUCTURES

CONDITIONAL STATEMENTS

```
if <expr>:
    <statement>
```

```
if <expr>:
    <statement>
    <statement>
    ...
    <statement>
<following_statement>
```



CONDITIONAL STATEMENTS

```
>>> x = 0
>>> y = 5

>>> if x < y:                                       # Truthy
...     print('yes')                                #output is yes

yes
>>> if y < x:                                       # Falsy
...     print('yes')

>>> if y < x or x < y:                              # Truthy
...     print('yes')                                #output is yes

>>> if y < x and x < y:                             # Falsy
...     print('yes')

>>> if 'aul' in 'grault':                           # Truthy
...     print('yes')                                #output is yes

```

CONDITIONAL STATEMENTS

```
if <expr>:
    <statement(s)>
else:
    <statement(s)>
```

```
>>> x = 20

>>> if x < 50:
...     print('(first suite)')
...     print('x is small')
... else:
...     print('(second suite)')
...     print('x is large')
...
(first suite)
x is small
```

CONDITIONAL STATEMENTS

```
if <expr>:
    <statement(s)>
elif <expr>:
    <statement(s)>
elif <expr>:
    <statement(s)>
    ...
else:
    <statement(s)>
```

```
>>> name = 'Joe'
>>> if name == 'Fred':
...     print('Hello Fred')
... elif name == 'Xander':
...     print('Hello Xander')
... elif name == 'Joe':
...     print('Hello Joe')
... elif name == 'Arnold':
...     print('Hello Arnold')
... else:
...     print("I don't know who you are!")
...
Hello Joe
```

CONDITIONAL STATEMENTS

```
if <expr>:
    <statement>
```

```
if <expr>: <statement>
```

```
if <expr>: <statement_1>; <statement_2>; ...; <statement_n>
```

```
>>> x = 2
>>> if x == 1: print('foo'); print('bar'); print('baz')
... elif x == 2: print('qux'); print('quux')
... else: print('corge'); print('grault')
...
qux
quux

>>> x = 3
>>> if x == 1: print('foo'); print('bar'); print('baz')
... elif x == 2: print('qux'); print('quux')
... else: print('corge'); print('grault')
...
corge
grault
```

CONDITIONAL STATEMENTS

```
if <expr>:
    <statement(s)>
else:
    <statement(s)>
```

```
<expr1> if <conditional_expr> else <expr2>
```

```
>>> m = a if a > b else b
```

```
>>> x = y = 40

>>> z = 1 + x if x > y else y + 2
>>> z
42

>>> z = (1 + x) if x > y else (y + 2)
>>> z
42
```

LOOP!

```
for i in <collection>
    <loop body>
```

```
for <var> in <iterable>:
    <statement(s)>
```

```
>>> for n in (0, 1, 2, 3):
...     print(n)
...
0
1
2
3
```

```
>>> for n in range(0, 4):
...     print(n)
...
0
1
2
3
```

LOOP!

```
>>> for n in range(0, 6, 2):
...     print(n)
...
0
2
4
```

```
>>> for n in range(4, 0, -1):
...     print(n)
...
4
3
2
1
```

LOOP!

```
while <expr>:
    <statement(s)>
```

```
>>> n = 5
>>> while n > 0:
...     n -= 1
...     print(n)
...
4
3
2
1
0
```

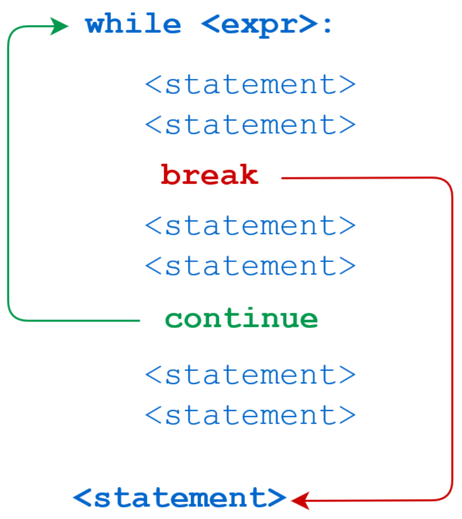
```
>>> n = 0
>>> while n > 0:
...     n -= 1
...     print(n)
...
```

LOOP!

```
while <expr>: <statement>
```

```
>>> n = 5
>>> while n > 0: n -= 1; print(n)

4
3
2
1
0
```

LOOP!



```
>>>n = 5
>>>while n > 0:
...     n -= 1
...     if n == 2:
...         break
...     print(n)
>>>print('Loop ended.')
4
3
Loop ended.
```

```
>>>n = 5
>>>while n > 0:
...     n -= 1
...     if n == 2:
...         continue
...     print(n)
>>>print('Loop ended.')
4
3
1
0
Loop ended.
```

LOOP!

```
while <expr>:
    <statement(s)>
else:
    <additional_statement(s)>
```

```
>>> n = 5
>>> while n > 0:
...     n -= 1
...     print(n)
... else:
...     print('Loop done.')
...
4
3
2
1
0
Loop done.
```

LOOP!

```
while <expr>:
    <statement(s)>
else:
    <additional_statement(s)>
```

```
>>> n = 5
>>> while n > 0:
...     n -= 1
...     print(n)
...     if n == 2:
...         break
... else:
...     print('Loop done.')
...
4
3
2
```

NESTED WHILE LOOPS!

```
while <expr1>:
    statement
    statement

    while <expr2>:
        statement
        statement
        break  # Applies to while <expr2>: loop

    break  # Applies to while <expr1>: loop
```

```
if <expr>:
    statement
    while <expr>:
        statement
        statement
else:
    while <expr>:
        statement
        statement
    statement
```

NESTED WHILE LOOPS!

```
while <expr>:
    if <expr>:
        statement
    elif <expr>:
        statement
    else:
        statement

    if <expr>:
        statement
```

EXAMLE

```
#printing the multiplication tables for the numbers 1 and 2.

# The outer loop
for i in range(1, 3):
    # The inner loop
    for j in range(1, 10):
        print(i, "*", j, "=", i*j)
    #newline to separate between each table.
    print()
"""
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9

2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
"""
```

EXAMLE

```python
#printing prime numbers between 2 and 99.

#since primes start from 2
i = 2
# Use a while loop to go through numbers from 2 to less than 100.
while i < 100:
    # For each 'i', initialize 'j' at 2.
    j = 2
    # Continue dividing until 'j' is greater than i divided by 'j'.
    while j <= (i/j):
        # If there is no remainder, 'i' is not prime, and we break out of the loop.
        if not(i % j):
            break
        # Increment 'j' by 1 to test the next potential factor.
        j = j + 1
    # If we've gone past the square root of 'i' without finding any factors,
    # then 'i' is a prime number.
    if j > i/j:
        print(i, "is prime")
    # Increment 'i' to check if the next number is prime.
    i = i + 1
# After checking all numbers print "Good bye!"
print("Good bye!")
```

EXAMLE

```python
#print a circle pattern

# Define the radius of the circle.
radius = 6

# Loop through a range from -radius to radius (inclusive) for the y-axis.
for y in range(-radius, radius + 1):
    # For each position on the y-axis, loop through the same range for the x-axis.
    for x in range(-radius, radius + 1):
        # Calculate the distance of the point (x, y) from the center (0, 0)
        distance = (x ** 2 + y ** 2) ** 0.5
        # If the distance is less than or equal to the radius, it is within the circle.
        if distance <= radius:
            # Print 'o' without moving to the next line.
            print("o", end="")
        else:
            # Print an space to represent a point outside the circle.
            print("  ", end="")

    # After printing all points on the current line, move to the next line.
    print()
```

FUNCTIONS

BUILT-IN FUNCTIONS

abs()	complex()	getattr()	len()	pow()	str()
all()	delattr()	globals()	list()	print()	sum()
any()	dict()	hasattr()	locals()	property()	super()
ascii()	dir()	hash()	map()	range()	tuple()
bin()	divmod()	help()	max()	repr()	type()
bool()	enumerate()	hex()	memoryview()	reversed()	vars()
bytearray()	eval()	id()	min()	round()	zip()
bytes()	exec()	input()	next()	set()	
callable()	filter()	int()	object()	setattr()	
chr()	float()	isinstance()	oct()	slice()	
classmethod()	format()	issubclass()	open()	sorted()	
compile()	frozenset()	iter()	ord()	staticmethod()	

BUILT-IN FUNCTIONS

```
>>>pow(2, 3)
8

>>>pow(2, 3, mod=3)
2
>>>2**3 % 3 == 2
True

>>>round(4.5)
4

>>>max(3, 4, 1)
4

>>len("hello")
5
```

USER-DEFINED FUNCTIONS

mathematical concept of a function

$$z = f(x,y)$$

```
def <function_name>([<parameters>]):
    <statement(s)>
```

USER-DEFINED FUNCTIONS

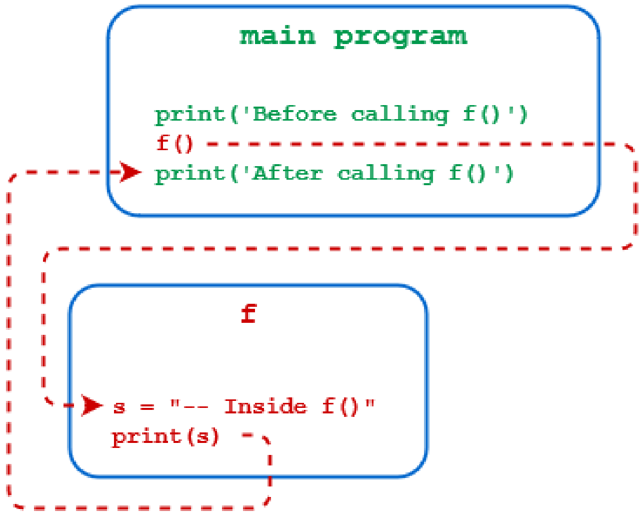
```
def <function_name>([<parameters>]):
    <statement(s)>
```

```
<function_name>([<arguments>])
```

Component	Meaning
def	The keyword that informs Python that a function is being defined
<function_name>	A valid Python identifier that names the function
<parameters>	An optional, comma-separated list of parameters that may be passed to the function
:	Punctuation that denotes the end of the Python function header (the name and parameter list)
<statement(s)>	A block of valid Python statements

USER-DEFINED FUNCTIONS

```python
def f():
    s = '-- Inside f()'
    print(s)

print('Before calling f()')
f()
print('After calling f()')
```



USER-DEFINED FUNCTIONS

```python
def call_name(name):
    print("hello", name)

call_name("ali") #output is hello ali
```

```python
def f(qty, item, price):
    print(qty, item, "cost $", price)

f(6, 'bananas', 1.74) #output is 6 bananas cost $ 1.74

f('bananas', 1.74, 6) #bananas 1.74 cost $ 6.00

# Too few arguments
f(6, 'bananas')
'''Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    f(6, 'bananas')
TypeError: f() missing 1 required positional argument: 'price'
'''
```

USER-DEFINED FUNCTIONS

```python
def f(qty, item, price):
    print(qty, item, "cost $", price)
```

```python
# Too few arguments
f(6, 'bananas')
'''Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    f(6, 'bananas')
TypeError: f() missing 1 required positional argument: 'price'
'''

# Too many arguments
f(6, 'bananas', 1.74, 'kumquats')
'''Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    f(6, 'bananas', 1.74, 'kumquats')
TypeError: f() takes 3 positional arguments but 4 were given
'''
```

USER-DEFINED FUNCTIONS

```python
def f(qty, item, price):
    print(qty, item, "cost $", price)
```

```python
#specify arguments
f(qty=6, item='bananas', price=1.74) #output is 6 bananas cost $1.74

f(item='bananas', price=1.74, qty=6) #output is 6 bananas cost $1.74

f(qty=6, item='bananas', cost=1.74)
'''Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: f() got an unexpected keyword argument 'cost'
'''
```

USER-DEFINED FUNCTIONS

```python
def f(qty, item, price):
    print(qty, item, "cost $", price)
```

```python
f(6, price=1.74, item='bananas') #output is 6 bananas cost $1.74

f(6, 'bananas', price=1.74) #output is 6 bananas cost $1.74

f(6, item='bananas', 1.74)
#SyntaxError: positional argument follows keyword argument
```

USER-DEFINED FUNCTIONS

```python
#Default Parameters
def f(qty=6, item='bananas', price=1.74):
    print(qty, item, "cost $", price)
```

```python
f(4, 'apples', 2.24) #output is 4 apples cost $2.24

f(4, 'apples') #output is 4 apples cost $1.74

f(4) #output is 4 bananas cost $1.74

f() #output is 6 bananas cost $1.74

f(item='kumquats', qty=9) #output is 9 kumquats cost $1.74

f(price=2.29) #output is 6 bananas cost $2.29
```

USER-DEFINED FUNCTIONS

```python
#The return Statement
def f():
    return 'foo'

s = f()
print(s) #output is 'foo'
```

```python
def f(x):
    if x < 100:
        return "small"
    if x > 100:
        return "big"

x = 65
p = f(x)
print(x, "is", p) #output is 65 is small
```

USER-DEFINED FUNCTIONS

```python
#The return Statement
def f():
    return 'foo', 'bar', 'baz', 'qux'


type(f()) #output is <class 'tuple'>
t = f()
print(t) #output is ('foo', 'bar', 'baz', 'qux')

a, b, c, d = f()
print("a =", a, "b =", b, "c =", c, "d =", d)
#output is a = foo, b = bar, c = baz, d = qux
```

USER-DEFINED FUNCTIONS

```python
def double(x):
    return x * 2


x = 5
x = double(x)
print(x) #output is 10
```

```python
def avg(a, b, c):
    return (a + b + c) / 3

print(avg(1, 2, 3)) #output is 2.0
```

EXAMPLE

```python
#printing prime numbers between 2 and 99.
def is_prime(number):
    """
    Checks if a given number is prime.
    """
    if number < 2:
        return False
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return False
    return True

def print_primes():
    """
    Prints prime numbers between 2 and 99.
    """
    for num in range(2, 100):
        if is_prime(num):
            print(num, "is prime")

print_primes()
print("Good bye!")
```

USER-DEFINED FUNCTIONS

```python
#Argument Tuple Packing
def f(*args):
    print(args)
    for x in args:
        print(x)
```

```python
f(1, 2, 3)
'''(1, 2, 3)
<class 'tuple'> 3
1
2
3
'''

f('foo', 'bar', 'baz', 'qux', 'quux')
'''('foo', 'bar', 'baz', 'qux', 'quux')
<class 'tuple'> 5
foo
bar
baz
qux
quux
'''
```

USER-DEFINED FUNCTIONS

```python
#Argument Tuple Packing
def avg(*args):
    total = 0
    for i in args:
        total += i
    return total / len(args)


print(avg(1, 2, 3)) #output is 2.0
print(avg(1, 2, 3, 4, 5)) #output is 3.0
```

```python
def avg(*args):
    return sum(args) / len(args)


print(avg(1, 2, 3)) #output is 2.0
print(avg(1, 2, 3, 4, 5)) #output is 3.0
```

USER-DEFINED FUNCTIONS

```python
#Argument Dictionary Packing
def f(**kwargs):
    print(kwargs)
    print(type(kwargs))
    for key, val in kwargs.items():
        print(key, '->', val)


f(foo=1, bar=2, baz=3)
'''{'foo': 1, 'bar': 2, 'baz': 3}
<class 'dict'>
foo -> 1
bar -> 2
baz -> 3
'''
```

USER-DEFINED FUNCTIONS

```python
#Argument Dictionary Packing
def f(a, b, *args, **kwargs):
    print(F'a = {a}')
    print(F'b = {b}')
    print(F'args = {args}')
    print(F'kwargs = {kwargs}')


f(1, 2, 'foo', 'bar', 'baz', 'qux', x=100, y=200, z=300)
'''a = 1
b = 2
args = ('foo', 'bar', 'baz', 'qux')
kwargs = {'x': 100, 'y': 200, 'z': 300}
'''
```

DATA STRUCTURES

DATA TYPES

- Text Type
  - str
- Numeric Types
  - int, float, complex
- Boolean Type
  - bool
- Void Type
  - None

```python
a = "Hello"             # str
b = 'Hello'             # str
c = str(10)             # str
d = 10                  # int
e = int(3.1)            # int
f = 3.14                # float
g = float('1.4')        # float
h = 2 + 3j              # complex
i = complex(2,3)        # complex
j = True                # bool
k = False               # bool
l = bool(1)             # bool
m = None                # NoneType
```

DATA TYPES

- Sequence Types
  - list, tuple, range
- Mapping Type
  - dict
- Set Types
  - set
- Binary Types
  - bytes

```python
a = [-1, "Text"]                # list
b = list([-1, 'Text'])          # list
c = (-1, "Text")                # tuple
d = tuple([-1, "Text"])         # tuple
e = range(1, 100, 2)            # range
f = {'e':2.71, 'pi': 3.14}      # dict
g = dict(name='ali', age=25)    # dict
h = {1,2,3,2}                   # set
i = set([1,2,3,2])              # set
```

STRINGS

```
>>> s = 'foo'
>>> t = 'bar'
>>> u = 'baz'

>>> s + t
'foobar'
>>> s + t + u
'foobarbaz'

>>> print('Go team' + '!!!')
Go team!!!
```

```
>>> s = 'foo.'

>>> s * 4
'foo.foo.foo.foo.'
>>> 4 * s
'foo.foo.foo.foo.'
>>> 'foo' * -8
''
```

STRINGS

```
>>> s = 'foo'

>>> s in 'That\'s food for thought.'
True
>>> s in 'That\'s good for now.'
False
```

```
>>> 'z' not in 'abc'
True
>>> 'z' not in 'xyz'
False
```

STRINGS

```
>>> s = 'I am a string.'
>>> len(s)
14
```

```
>>> str(49.2)
'49.2'
>>> str(3+4j)
'(3+4j)'
>>> str(3 + 29)
'32'
>>> str('foo')
'foo'
```

STRINGS INDEXING

```
>>> s = 'foobar'

>>> s[0]
'f'
>>> s[1]
'o'
>>> s[3]
'b'
>>> len(s)
6
>>> s[len(s)-1]
'r'
>>> s = 'foobar'
>>> s[-1]
'r'
>>> s[-2]
'a'
>>> len(s)
6
>>> s[-len(s)]
'f'
```

STRINGS INDEXING

```
>>> s = 'foobar'
>>> s[2:5]
'oba'
>>> s = 'foobar'
>>> s[:4]
'foob'
>>> s[0:4]
'foob'
>>> s = 'foobar'
>>> s[2:]
'obar'
>>> s[2:len(s)]
'obar'
>>> s = 'foobar'
>>> s[:4] + s[4:]
'foobar'
>>> s[:4] + s[4:] == s
True
>>> s = 'foobar'
>>> t = s[:]
>>> s is t
True
```

STRINGS INDEXING

```
>>> s = 'foobar'

>>> s[0:6:2]
'foa'

>>> s[1:6:2]
'obr'

>>> s = '12345' * 5
>>> s
'1234512345123451234512345'
>>> s[::5]
'11111'
>>> s[4::5]
'55555'
>>> s = '12345' * 5
>>> s
'1234512345123451234512345'
>>> s[::-5]
'55555'
>>> s = 'If Comrade Napoleon says it, it must be right.'
>>> s[::-1]
'.thgir eb tsum ti ,ti syas noelopaN edarmoC fI'
```

STRINGS

```
>> 'hello ali'.split()
['hello', 'ali']

>> 'Hello ali'.replace('Hello', 'Bye')
'Bye ali'

>> '-'.join(['a', 'b', 'c'])
'a-b-c'

>> 'Hello'.upper()
'HELLO'

>> 'Hello'.lower()
'hello'

>>> 'foo bar foo baz foo qux'.find('foo')
0
>>> 'foo bar foo baz foo qux'.find('foo', 4)
8
```

STRINGS

Method	Description
capitalize()	Converts the first character to upper case
count()	Returns the number of times a specified value occurs in a string
endswith()	Returns true if the string ends with the specified value
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
join()	Converts the elements of an iterable into a string
lower()	Converts a string into lower case
replace()	Returns a string where a specified value is replaced with a specified value
split()	Splits the string at the specified separator, and returns a list

LISTS

```
>>> colors = [
...       "red",
...       "orange",
...       "yellow",
...       "green",
...       "blue",
...       "indigo",
...       "violet"
... ]

>>> colors
['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
```

LISTS

```
>>> colors[0]
'red'
>>> colors[1]
'orange'
>>> colors[2]
'yellow'
>>> colors[3]
'green'

>>> colors[-1]
'green'
>>> colors[-2]
'yellow'
>>> colors[-3]
'orange'
>>> colors[-4]
'red'

>>> languages[-7]
Traceback (most recent call last):
    ...
IndexError: list index out of range
```

LISTS

```
>>> colors[0:2]
['red', 'orange']
>>> colors[1:]
['orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
>>> colors[0::2]
['red', 'yellow', 'blue', 'violet']
>>> colors[0::2]
['red', 'yellow', 'blue', 'violet']
>>> colors[-8::1]
['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
```

LISTS

```
>>> countries = ["United States", "Canada", "Poland", "Germany", "Austria"]

>>> nations = countries
>>> id(countries) == id(nations)
True

>>> nations = countries[:]
>>> nations
['United States', 'Canada', 'Poland', 'Germany', 'Austria']

>>> id(countries) == id(nations)
False

>>>from copy import copy
>>> nations = countries.copy()
>>> nations
['United States', 'Canada', 'Poland', 'Germany', 'Austria']

>>> id(countries) == id(nations)
False
```

LISTS

```
>>> pets = ["cat", "dog"]

>>> pets.append("parrot")
['cat', 'dog', 'parrot']

>>> pets.append(['hamster', 'turtle'])
['cat', 'dog', 'parrot', ['hamster', 'turtle']]


>>> pets.extend(['hamster', 'turtle'])
['cat', 'dog', 'parrot', 'hamster', 'turtle']

>>> pets.insert(2, 'hamster')
['cat', 'dog', 'hamster', 'parrot', 'hamster', 'turtle']

>>> pets.remove('hamster')
['cat', 'dog', 'parrot', 'hamster', 'turtle']

>>> visited = pets.pop()
>>> visited
'turtle'
>>> pets
['cat', 'dog', 'parrot', 'hamster']
```

LISTS

```
>>> numbers = ["2", "9", "5", "1", "6"]

>>> for i, number in enumerate(numbers):
...     numbers[i] = int(number)
...

>>> numbers
[2, 9, 5, 1, 6]
```

LISTS

Method	**Description**
append()	Adds an element at the end of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

TUPLES

```
>>> record = ("John", 35, "Python Developer")

>>> record[0] = "Adel"
TypeError: 'tuple' object does not support item assignment

>>> record[0]
'John'
>>> record[1]
35
>>> record[2]
'Python Developer'

>>> record[:2]
("John", 3)

>>> record[1:]
(35, "Python Developer")
```

TUPLES

```
>>> point = (7, 14, 21)

>>> x, y, z = point
>>> x
7
>>> y
14
>>> z
21
```

TUPLES

```
>>> student_info = ("Linda", 18, ["Math", "Physics", "History"])

>>> student_profile = student_info[:]
>>> id(student_info) == id(student_profile)
True

>>> from copy import copy
>>> student_info = ("Linda", 18, ["Math", "Physics", "History"])
>>> student_profile = copy(student_info)
>>> id(student_info) == id(student_profile)
True
```

SETS

```
>>> x = set(['foo', 'bar', 'baz', 'foo', 'qux'])
>>> x
{'qux', 'foo', 'bar', 'baz'}

>>> x = set(('foo', 'bar', 'baz', 'foo'))
>>> x
{'foo', 'bar', 'baz'}

>>> len(x)
3

>>> 'bar' in x
True
>>> 'qux' in x
False
```

SETS

```
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'baz', 'qux', 'quux'}
>>> x1 | x2
{'baz', 'quux', 'qux', 'bar', 'foo'}

>>> x1.union(x2)
{'baz', 'quux', 'qux', 'bar', 'foo'}

>>> a = {1, 2, 3, 4}
>>> b = {2, 3, 4, 5}
>>> c = {3, 4, 5, 6}
>>> d = {4, 5, 6, 7}

>>> a.union(b, c, d)
{1, 2, 3, 4, 5, 6, 7}

>>> a | b | c | d
{1, 2, 3, 4, 5, 6, 7}
```

SETS

```
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'baz', 'qux', 'quux'}

>>> x1.intersection(x2)
{'baz'}

>>> x1 & x2
{'baz'}

>>> a = {1, 2, 3, 4}
>>> b = {2, 3, 4, 5}
>>> c = {3, 4, 5, 6}
>>> d = {4, 5, 6, 7}

>>> a.intersection(b, c, d)
{4}

>>> a & b & c & d
{4}
```

SETS

```
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'baz', 'qux', 'quux'}

>>> x1.difference(x2)
{'foo', 'bar'}

>>> x1 - x2
{'foo', 'bar'}

>>> a = {1, 2, 3, 30, 300}
>>> b = {10, 20, 30, 40}
>>> c = {100, 200, 300, 400}

>>> a.difference(b, c)
{1, 2, 3}

>>> a - b - c
{1, 2, 3}
```

SETS

```
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'baz', 'qux', 'quux'}

>>> x1.symmetric_difference(x2)
{'foo', 'qux', 'quux', 'bar'}

>>> x1 ^ x2
{'foo', 'qux', 'quux', 'bar'}

>>> a = {1, 2, 3, 4, 5}
>>> b = {10, 2, 3, 4, 50}
>>> c = {1, 50, 100}

>>> a ^ b ^ c
{100, 5, 10}
```

SETS

```
>>> x1 = {1, 3, 5}
>>> x2 = {2, 4, 6}

>>> x1.isdisjoint(x2)
True
>>> x1 & x2
set()

>>> x1 = {1, 3, 5}
>>> x2 = {1, 2, 3, 4, 5}
>>> x1.issubset(x2)
True

>>> x1 <= x2
True

>>> x2.issuperset(x1)
True

>>> x2 >= x1
True
```

SETS

```
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'foo', 'baz', 'qux'}

>>> x1 |= x2
>>> x1
{'qux', 'foo', 'bar', 'baz'}

>>> x1.update(['corge', 'garply'])
>>> x1
{'qux', 'corge', 'garply', 'foo', 'bar', 'baz'}


>>> x1 &= x2
>>> x1
{'foo', 'baz'}

>>> x1.intersection_update(['baz', 'qux'])
>>> x1
{'baz'}
```

SETS

```
>>> x = {'foo', 'bar', 'baz'}

>>> x.discard('baz')
>>> x
{'bar', 'foo'}

>>> x.discard('qux')
>>> x
{'bar', 'foo'}

>>> x.pop()
'bar'
>>> x
{'baz', 'foo'}
```

SETS

Method	**Description**
add()	Adds an element to the set
difference()	Returns a set containing the difference between two or more sets
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two or more sets
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
symmetric_difference()	Returns a set with the symmetric differences of two sets
union()	Return a set containing the union of sets

DICTIONARIES

```
d = {
    <key>: <value>,
    <key>: <value>,
      .
      .
      .
    <key>: <value>
}
```

```
>>> MLB_team = {
...     'Colorado' : 'Rockies',
...     'Boston'   : 'Red Sox',
...     'Minnesota': 'Twins',
...     'Milwaukee': 'Brewers',
...     'Seattle'  : 'Mariners'
... }

>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins',
'Milwaukee': 'Brewers', 'Seattle': 'Mariners'}
```

DICTIONARIES

```
>>> MLB_team['Minnesota']
'Twins'
>>> MLB_team['Colorado']
'Rockies'

>>> MLB_team['Kansas City'] = 'Royals'
>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins',
'Milwaukee': 'Brewers', 'Seattle': 'Mariners', 'Kansas City': 'Royals'}

>>> MLB_team['Seattle'] = 'Seahawks'
>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins',
'Milwaukee': 'Brewers', 'Seattle': 'Seahawks', 'Kansas City': 'Royals'}

>>> del MLB_team['Seattle']
>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins',
'Milwaukee': 'Brewers', 'Kansas City': 'Royals'}
```

DICTIONARIES

```
>>> person = {}
>>> type(person)
<class 'dict'>

>>> person['fname'] = 'Joe'
>>> person['lname'] = 'Fonebone'
>>> person['age'] = 51
>>> person['spouse'] = 'Edna'
>>> person['children'] = ['Ralph', 'Betty', 'Joey']
>>> person['pets'] = {'dog': 'Fido', 'cat': 'Sox'}

>>> person
{'fname': 'Joe', 'lname': 'Fonebone', 'age': 51, 'spouse': 'Edna',
'children': ['Ralph', 'Betty', 'Joey'], 'pets': {'dog': 'Fido', 'cat': 'Sox'}}

>>> person['fname']
'Joe'
>>> person['age']
51
>>> person['children']
['Ralph', 'Betty', 'Joey']
```

DICTIONARIES

```
>>> person
{'fname': 'Joe', 'lname': 'Fonebone', 'age': 51, 'spouse': 'Edna',
'children': ['Ralph', 'Betty', 'Joey'], 'pets': {'dog': 'Fido', 'cat': 'Sox'}}

>>> person['children'][-1]
'Joey'
>>> person['pets']['cat']
'Sox'
```

DICTIONARIES

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}

>>> list(d.items())
[('a', 10), ('b', 20), ('c', 30)]
>>> list(d.items())[1][0]
'b'
>>> list(d.items())[1][1]
20
```

DICTIONARIES

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}

>>> list(d.keys())
['a', 'b', 'c']

>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}

>>> list(d.values())
[10, 20, 30]
```

DICTIONARIES

```
>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> d.pop('b')
20
>>> d
{'a': 10, 'c': 30}

>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> d.popitem()
('c', 30)
>>> d
{'a': 10, 'b': 20}

>>> d.popitem()
('b', 20)
>>> d
{'a': 10}
```

DICTIONARIES

```
>>> d1 = {'a': 10, 'b': 20, 'c': 30}
>>> d2 = {'b': 200, 'd': 400}

>>> d1.update(d2)
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}

>>> d1 = {'a': 10, 'b': 20, 'c': 30}
>>> d1.update(b=200, d=400)
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

DICTIONARIES

```python
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

for x in thisdict:
    print(x)
'''brand
model
year'''


for x in thisdict:
  print(thisdict[x])

'''Ford
Mustang
1964'''
```

DICTIONARIES

```python
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

for x in thisdict:
    print(x)
'''brand
model
year'''


for x in thisdict:
  print(thisdict[x])

'''Ford
Mustang
1964'''
```

DICTIONARIES

```python
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

for x in thisdict.keys():
  print(x)
'''brand
model
year'''


for x in thisdict.values():
  print(x)
'''Ford
Mustang
1964'''

for x, y in thisdict.items():
  print(x, y)
'''brand Ford
model Mustang
year 1964'''
```

LIBRARIES

IMPORT LIBRARY

```python
import math

print(math.cos(0))
```

```python
import math as m

print(m.cos(0))
```

```python
from math import cos

print(cos(0))
```

```python
from math import *

print(cos(0))
```

INSTALL LIBRARY

```
pip install numpy
```

```
pip install —upgrade numpy
```

```
pip install numpy==1.23.5
```

```
conda install numpy
```

```
conda install numpy=1.13
```

FILES

FILES I/O

```python
f = open('file name', 'mode')
f.close()
```

Mode	Description
'x'	Exclusive creation that fails if the file already exists.
'w'	Writes to a file and creates the file if it does not exist or overwrites an existing file.
'r'	Reads from a file and returns an error if the file does not exist (default).
'a'	Appends to a file and creates the file if it does not exist or overwrites an existing file.

FILES I/O

```python
f = open('file.txt', 'x') #create new txt file
f.close()
```

```python
f = open('file.txt', 'x') #If the file already exists
f.close()

#FileExistsError: [Errno 17] File exists: 'file.txt'
```

WRITE MODE

```
f = open('file.txt', 'w') #open file in a write mode
f.close()
```

```
f = open('file.txt', 'w')
f.write("hello") #output is a txt file contains "hello"
f.close()
```

```
f = open('file.txt', 'w')
f.write("Bye") #output is a txt file contains "Bye"
#It doesn't contain "hello"
f.close()
```

WRITE MODE

```python
f = open('file.txt', 'w') #open file in a write mode
for i in range(10):
  f.write("this is line %d .\n" %i)
f.close()

"""
this is line 0 .
this is line 1 .
this is line 2 .
this is line 3 .
this is line 4 .
this is line 5 .
this is line 6 .
this is line 7 .
this is line 8 .
this is line 9 .

"""
```

WRITE MODE

```python
f = open('file.txt', 'w')
print(f.tell()) #curent position in the file
f.seek(0) #move to the first character
f.seek(10) #move to the 10th character
```

```python
f = open('file.txt', 'w')
f.write("hello")
f.seek(0)
f.write("bye")

#output is byelo
```

APPEND MODE

```python
f = open('file.txt', 'a') #open file in a append mode
f.close()
```

```python
f = open('file.txt', 'a')
f.write("hello") #output is a txt file contains "hello"
f.close()
```

```python
f = open('file.txt', 'a')
f.write("Bye") #output is a txt file contains "helloBye"
f.close()
```

READ MODE

```python
f = open('file.txt', 'r') #open file in a read mode
f.close()
```

```python
f = open('file.txt', 'r')
f.read() #reads all characters
f.close()
```

```python
f = open('file.txt', 'r')
for item in f.read():
  print(item)
```

```python
f = open('file.txt', 'r')
print(f.read())
```

READ MODE

```python
f = open('file.txt', 'r')
print(f.read(10))
```

```python
f = open('file.txt', 'r')
for item in f.readline():
  print(item)
```

```python
f = open('file.txt', 'r')
print(f.readline())
```

READ MODE

```python
f = open('file.txt', 'r')
for item in f.readlines():
  print(item)
```

```python
f = open('file.txt', 'r')
print(f.readlines())
```

READ MODE

```python
f = open('file.txt', 'r') #for large files
for item in f:
  print(item, end= "")
```

```python
f = open('file.txt', 'r')
batch = 10
f_batch= f.read(batch)
while len(f_batch) > 0:
  print(f_batch, end= "")
  f_batch = f.read(batch)
```

CONTEXT MANAGERS

```python
with open('data.txt', 'r') as f:
    data = f.read()
```

```python
with open('data.txt', 'w') as f:
    data = 'some data to be written to the file'
    f.write(data)
```

CONTEXT MANAGERS

```python
with open('file.txt', 'r') as fr:
    with open('file2.txt', 'w') as fw:
        for line in fr:
            fw.write(line)
```

```python
batch = 10
with open('file.txt', 'r') as fr:
    with open('file2.txt', 'w') as fw:
      fr_batch = fr.read(batch)
      while len(fr_batch) > 0:
        fw.write(fr_batch)
        fr_batch = fr.read(batch)

```

EXCEPTION IN PYTHON

RAISING AN EXCEPTION

```python
number = 10
if number > 5:
    raise Exception(f"The number should not exceed 5. ({number=})")
print(number)

"""
Traceback (most recent call last):
  File "./low.py", line 3, in <module>
    raise Exception(f"The number should not exceed 5. ({number=})")
Exception: The number should not exceed 5. (number=10)
"""
```

TRY AND EXCEPT BLOCK

```python
def linux_interaction():
    import sys
    if "linux" not in sys.platform:
        raise RuntimeError("Function can only run on Linux systems.")
    print("Doing Linux things.")
```

```python
try:
    linux_interaction()
except:
    pass
```

TRY AND EXCEPT BLOCK

```python
def linux_interaction():
    import sys
    if "linux" not in sys.platform:
        raise RuntimeError("Function can only run on Linux systems.")
    print("Doing Linux things.")
```

```python
try:
    linux_interaction()
except:
    print("Linux function wasn't executed.")
```

TRY AND EXCEPT BLOCK

```python
def linux_interaction():
    import sys
    if "linux" not in sys.platform:
        raise RuntimeError("Function can only run on Linux systems.")
    print("Doing Linux things.")
```

```python
try:
    linux_interaction()
except RuntimeError as error:
    print(error)
    print("The linux_interaction() function wasn't executed.")
```

TRY AND EXCEPT BLOCK

```python
try:
    with open("file.log") as file:
        read_data = file.read()
except:
    print("Couldn't open file.log")
```

```python
try:
    with open("file.log") as file:
        read_data = file.read()
except FileNotFoundError as fnf_error:
    print(fnf_error)
```

TRY AND EXCEPT BLOCK

```python
try:
    linux_interaction()
    with open("file.log") as file:
        read_data = file.read()
except FileNotFoundError as fnf_error:
    print(fnf_error)
except RuntimeError as error:
    print(error)
    print("Linux linux_interaction() function wasn't executed.")
```

TRY AND EXCEPT BLOCK

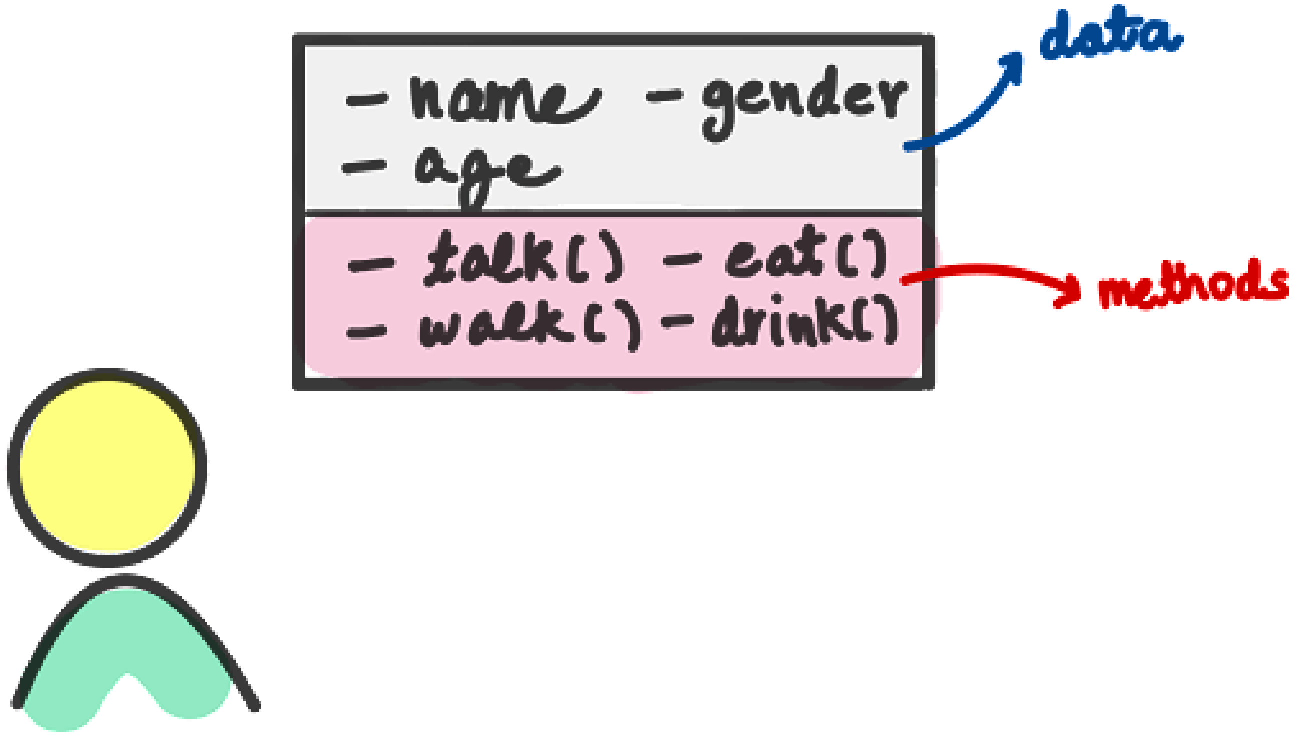
```python
try:
    linux_interaction()
except RuntimeError as error:
    print(error)
else:
    print("Doing even more Linux things.")
```

```python
try:
    linux_interaction()
except RuntimeError as error:
    print(error)

print("Doing even more Linux things.")
```

TRY AND EXCEPT BLOCK

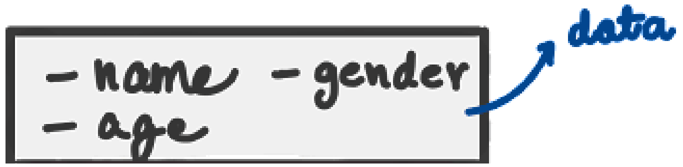
```python
try:
    linux_interaction()
except RuntimeError as error:
    print(error)
else:
    try:
        with open("file.log") as file:
            read_data = file.read()
    except FileNotFoundError as fnf_error:
        print(fnf_error)
```

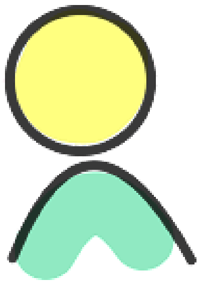
OBJECT-ORIENTED PROGRAMMING


- name
- gender
- age
- talk()
- eat()
- walk()
- drink()
data
methods

CREATE CLASS

```python
class human:
    name = ""
    age = 0
    gender = None
```





CREATE OBJECT

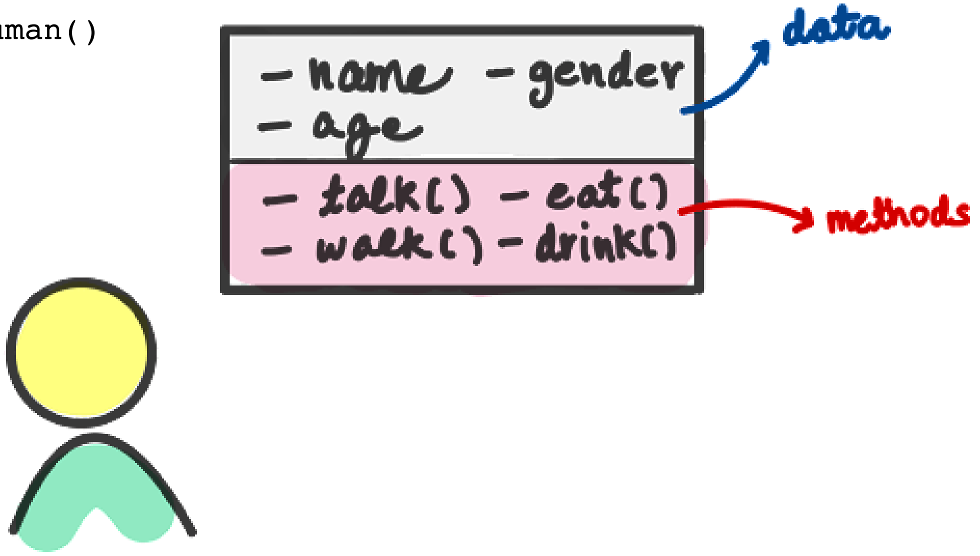
```python
class human:
    name = ""
    age = 0
    gender = None


ali = human()
zahra = human()
hadis = human()

print(type(ali)) #<class '__main__.human'>
print(type(zahra)) #<class '__main__.human'>
print(type(hadis)) #<class '__main__.human'>
```

CREATE OBJECT

```python
class human:
    name = ""
    age = 0
    gender = None


ali = human()
ali.name = "ali"
ali.age = 20
ali.gender = "M"

print(ali.name) #ali
print(ali.age) #20
print(ali.gender) #M
```

CREATE OBJECT

```
class human:
    name = ""
    age = 0
    gender = None

    def talk(self):
      print("Hi i'm %s" %self.name)

ali =human()
```



CREATE OBJECT

```python
class human:
    name = ""
    age = 0
    gender = None

    def talk(self):
      print("Hi i'm %s" %self.name)

    def walk(self):
        print("%s is walking" %self.name)
    def eat(self):
       print("%s is eating" %self.name)

    def dink(self):
       print("%s is drinking" %self.name)

ali =human()
```

CREATE OBJECT

```python
class human:
    name = ""
    age = 0
    gender = None

    def talk(self):
      print("Hi i'm %s" %self.name)

    def walk(self):
        print("%s is walking" %self.name)
    def eat(self):
       print("%s is eating" %self.name)

    def dink(self):
       print("%s is drinking" %self.name)

ali =human()
ali.name = "ali"
ali.talk() #Hi i'm ali
```

CREATE OBJECT

```python
class human:
    name = ""
    age = 0
    gender = None

    def talk(self):
      print("Hi i'm %s" %self.name)

    def walk(self, place):
        print("%s is walking in %s" %(self.name, place))


ali =human()
ali.name = "ali"
ali.walk("street") #ali is walking in street
```

INITIALIZE OBJECT

```python
class human:

    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender

ali = human("ali", 20, "M")
print(type(ali)) #<class '__main__.human'>
print(ali.name) #ali
```

INITIALIZE OBJECT

```python
class human:

    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender

    def talk(self):
      print("Hi i'm %s" %self.name)


ali =human()
ali.name = "ali"
ali.talk() #Hi i'm ali
```

INITIALIZE OBJECT

```python
class Human:
    def __init__(self, name="", age=0, gender=None):
        self.name = name
        self.age = age
        self.gender = gender

    def walk(self, place):
        print("%s is walking in %s" % (self.name, place))

ali = Human(name="Ali", gender="Male")
print(ali.age) #0
ali.walk("Home") #Ali is walking in Home
```

DESTRUCT OBJECT

```python
class Human:
    def __init__(self, name="", age=0, gender=None):
        self.name = name
        self.age = age
        self.gender = gender

    def __del__(self):
        print("%s has been deleted" % self.name)

ali = Human("Ali", 20, "M")
del ali #Ali has been deleted
print(ali.name) #NameError: name 'ali' is not defined
```

REPRESENT OBJECT

```python
class Human:
    def __init__(self, name="", age=0, gender=None):
        self.name = name
        self.age = age
        self.gender = gender


ali = Human("Ali", 20, "M")
print(ali) #<__main__.Human object at 0x000001AD05851AD0>
```

REPRESENT OBJECT

```python
class Human:
    def __init__(self, name="", age=0, gender=None):
        self.name = name
        self.age = age
        self.gender = gender

    def __repr__(self):
        return f"hello {self.name}"


ali = Human("Ali", 20, "M")
print(ali) #hello Ali
```

CREATE OBJECTS

```python
class User:
    users = []

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        User.users.append(self.email)



maryam = User(name="Maryam", password="secret", email="maryam@example.com")
ali =  User(name="ali", password="secret", email="ali@example.com")
```

CREATE OBJECTS

```python
class User:
    users = []

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        User.users.append(self.email)



maryam = User(name="Maryam", password="secret", email="maryam@example.com")
ali =  User(name="ali", password="secret", email="ali@example.com")

print(ali.users) #['maryam@example.com', 'ali@example.com']
```

” CREATE OBJECTS

```python
class User:
    users = []

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        User.users.append(self.email)

    def login(self, email, password):
        if email in User.users:
            if self.password == password:
                print("Welcome!")
            else:
                print("Wrong password")
        else:
            print("Please sign up first")


maryam = User(name="Maryam", password="secret", email="maryam@example.com")
maryam.login(email="maryam@example.com", password="secret")
```

CREATE OBJECTS

```python
class User:
    users = []

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        User.users.append(self.email)

    def change_pass(self, old_pass, new_pass):
        if old_pass != self.password:
            print("wrong old password")
        else:
            self.password = new_pass



maryam = User(name="Maryam", password="secret", email="maryam@example.com")
print(maryam.password) #secret
maryam.change_pass("secret", "newsecret")
print(maryam.password) #secret
```

CREATE OBJECTS

```python
class User:
    users = []

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        User.users.append(self.email)

    def __del__(self):
        User.users.remove(self.email)



maryam = User(name="Maryam", password="secret", email="maryam@example.com")
ali = User(name="ali", password="secret", email="ali@example.com")

print(ali.users) #['maryam@example.com', 'ali@example.com']
del maryam
print(ali.users) #['ali@example.com']
```

”CREATE OBJECTS

```python
class User:
    users = []

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        self.followers = []
        self.followings = []
        User.users.append(self.email)

    def follow(self, user):
        self.followings.append(user.name)
        user.followers.append(self.name)

    def unfollow(self, user):
        self.followings.remove(user.name)
        user.followers.remove(self.name)


maryam = User(name="Maryam", password="secret", email="maryam@example.com")
ali = User(name="ali", password="secret", email="ali@example.com")
maryam.follow(ali)
print(maryam.followings) #['ali']
maryam.unfollow(ali)
print(maryam.followings) #[]

```

CREATE OBJECTS

```python
class User:
    users = []
    all_followings = {}

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        self.followers = []
        self.followings = []
        User.users.append(self.email)
        User.all_followings[self.name] = []

    def follow(self, user):
        self.followings.append(user.name)
        user.followers.append(self.name)
        User.all_followings[self.name].append(user.name)


maryam = User(name="Maryam", password="secret", email="maryam@example.com")
ali = User(name="ali", password="secret", email="ali@example.com")
maryam.follow(ali)
print(maryam.followings) #['ali']
print(maryam.all_followings) #{'Maryam': ['ali'], 'ali': []}

```

CREATE OBJECTS

```python
class User:
    users = []
    all_followings = {}

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        self.followers = []
        self.followings = []
        User.users.append(self.email)
        User.all_followings[self.name] = []

    def follow(self, user):
        self.followings.append(user.name)
        user.followers.append(self.name)
        User.all_followings[self.name].append(user.name)

    def unfollow(self, user):
        self.followings.remove(user.name)
        user.followers.remove(self.name)
        User.all_followings[self.name].remove(user.name)
```

CREATE OBJECTS

```python
class User:
    users = []
    all_followings = {}

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        self.followers = []
        self.followings = []
        User.users.append(self.email)
        User.all_followings[self.name] = []

    def follow(self, user):
        self.followings.append(user.name)
        user.followers.append(self.name)
        User.all_followings[self.name].append(user.name)

    def unfollow(self, user):
        self.followings.remove(user.name)
        user.followers.remove(self.name)
        User.all_followings[self.name].remove(user.name)

maryam = User(name="Maryam", password="secret", email="maryam@example.com")
ali = User(name="ali", password="secret", email="ali@example.com")
maryam.follow(ali)
print(maryam.followings) #['ali']
print(maryam.all_followings["Maryam"]) #['ali']
```

CREATE OBJECTS

```python
class User:
    users = []
    all_followings = {}

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        self.followers = []
        self.followings = []
        User.users.append(self)
        User.all_followings[self.name] = []

    def __del__(self):
        for i in User.users:
            if i != self:
                self.unfollow(i)
        User.users.remove(self)

    def unfollow(self, person):
        self.followings.remove(person.name)
        person.followers.remove(self.name)
        User.all_followings[self.name].remove(person.name)
```

CREATE OBJECTS

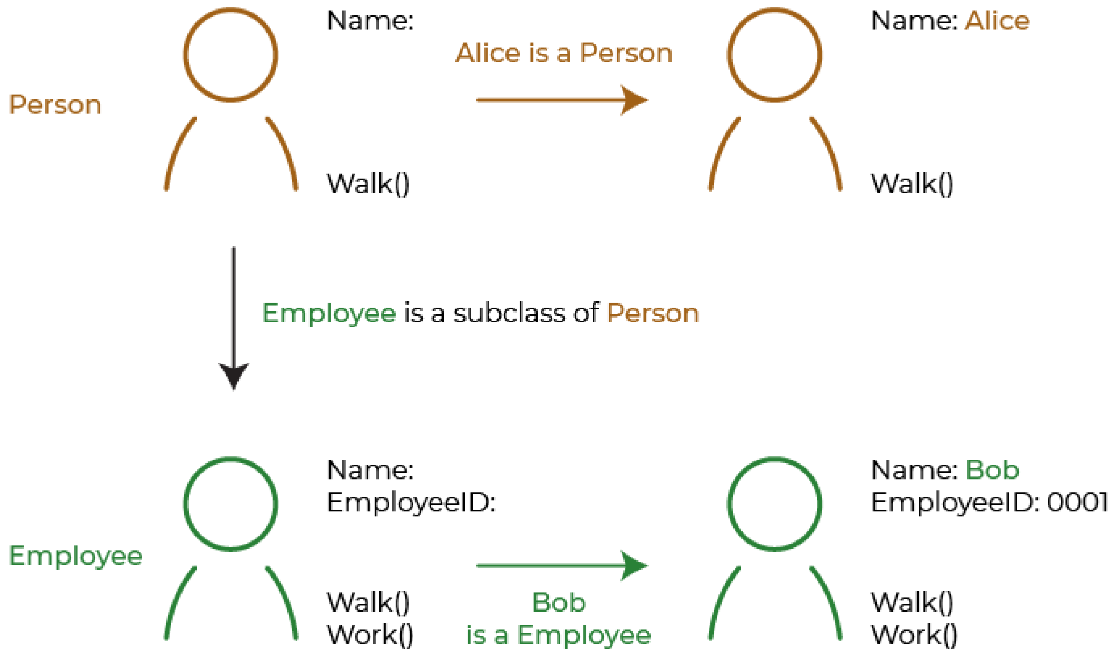
```python
class User:
    users = []
    all_followings = {}

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        self.followers = []
        self.followings = []
        User.users.append(self)
        User.all_followings[self.name] = []

    def remove_account(self):
        for i in User.users:
            if i != self:
                self.unfollow(i)
        User.users.remove(self)

    def unfollow(self, person):
        self.followings.remove(person.name)
        person.followers.remove(self.name)
        User.all_followings[self.name].remove(person.name)
```

CREATE OBJECTS

If you are interested:

```python
class User:
    users = []
    all_followings = {}

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        self.followers = []
        self.followings = []
        User.users.append(self)
        User.all_followings[self.name] = []

    def remove_account(self):
        for person_name in self.followings[:]:
            person = next((user for user in User.users if user.name == person_name), None)
            if person:
                self.unfollow(person)

        for follower in self.followers[:]:
            follower.unfollow(self)

        User.users.remove(self)
        del User.all_followings[self.name]


    def follow(self, person):
        self.followings.append(person)
        person.followers.append(self)
        User.all_followings[self.name].append(person.name)

    def unfollow(self, person):
        self.followings.remove(person)
        person.followers.remove(self)
        User.all_followings[self.name].remove(person.name)
```

INHERITANCE



INHERITANCE

```python
class human:
    def __init__(self, name=""):
       self.name = name

    def walk(self):
        print("%s is walking" %self.name)


class employee(human):
  pass

ali =human("Ali")
ali.walk() #Ali is walking
maryam = employee("Maryam")
maryam.walk() #Maryam is walking
```

INHERITANCE

```python
class human:
    def __init__(self, name=""):
       self.name = name

    def walk(self):
        print("%s is walking" %self.name)


class employee(human):
  def __init__(self, name="", EmployeeID=000):
     human.__init__(self,name)
     self.EmployeeID = EmployeeID


ali =human("Ali")
maryam = employee("Maryam", 457)
print(maryam.EmployeeID) #457
print(ali.EmployeeID)
#AttributeError: 'human' object has no attribute 'EmployeeID'
```

INHERITANCE

```
class human:
    def __init__(self, name=""):
       self.name = name

    def walk(self):
        print("%s is walking" %self.name)


class employee(human):
  def __init__(self, name="", EmployeeID=000):
     super().__init__(name)
     self.EmployeeID = EmployeeID


ali =human("Ali")
maryam = employee("Maryam", 457)
print(maryam.EmployeeID) #457
print(ali.EmployeeID)
#AttributeError: 'human' object has no attribute 'EmployeeID'
```

INHERITANCE

```python
class human:
    def __init__(self, name="", age=0, gender=""):
       self.name = name
       self.age = age
       self.gender = gender

    def walk(self):
        print("%s is walking" %self.name)


class employee(human):
  def __init__(self, name="", age=0, gender="", EmployeeID=000):
     super().__init__(name, age, gender)
     self.EmployeeID = EmployeeID


ali =human("Ali", 19, "M")
maryam = employee("Maryam",20, "F", 457)
```

INHERITANCE

```
class human:
    def __init__(self, name="", age=0, gender=""):
       self.name = name
       self.age = age
       self.gender = gender

    def walk(self):
        print("%s is walking" %self.name)


class employee(human):
  def __init__(self, EmployeeID=000, **kwargs):
     super().__init__(**kwargs)
     self.EmployeeID = EmployeeID


ali =human("Ali", 19, "M")
maryam = employee(name="Maryam",age=20, gender="F", EmployeeID=457)
```

INHERITANCE

```python
class human:
    def __init__(self, name=""):
       self.name = name

    def walk(self):
        print("%s is walking" %self.name)


class employee(human):
  def __init__(self, name="", EmployeeID=000):
     super().__init__(name)
     self.EmployeeID = EmployeeID

  def work(self):
      print("%s is working" %self.name)


ali =human("Ali")
maryam = employee("Maryam", 457)
maryam.work() #Maryam is working
ali.work() #'human' object has no attribute 'work'
```

OVERWRITING

```python
class parent:
    def func(self):
        print("Hello Parent")

class child(parent1):
    def func(self):
        print("Hello Child")


ali = child()
reza = parent()
reza.func() #Hello Parent
ali.func() #Hello Child
```

MULTIPLE INHERITANCE

```python
class parent1:
    def func1(self):
        print("Hello Parent1")

class parent2:
    def func2(self):
        print("Hello Parent2")

class parent3:
    def func2(self):
        print("Hello Parent3")
    def func3(self):
        print("Hello Parent3_2")

class child(parent1, parent2, parent3):
    def func3(self):
        print("Hello Child")

    def func4(self):
        print("Hello Child2")

test = child()
test.func1() #Hello Parent1
test.func2() #Hello Parent2
test.func3()  #Hello Child
test.func4()  #Hello Child2
```

POLYMORPHISM

```python
len("hi") #a method in str class

len([2, 3]) #a method in list class
```

```python
class parent:
    def func(self):
        print("Hello Parent")

class child(parent1):
    def func(self):
        print("Hello Child")


ali = child()
reza = parent()
reza.func() #Hello Parent
ali.func() #Hello Child
```

POLYMORPHISM

```python
class Car:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Drive!")

class Boat:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Sail!")


car1 = Car("Ford", "Mustang")         #Create a Car class
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat class

car1.move() #Drive!
boat1.move() #Sail!
```

DECORATOR

```python
#function
def talk():
  print("hello")
```

```python
#Inner function
def talk():
  print("hello")

  def child_talk():
    print("inner hello")


talk() #hello
```

```python
#Inner function
def talk():
  print("hello")

  def child_talk():
    print("inner hello")

  return child_talk()


talk() #hello \n inner hello
```

DECORATOR

```python
#Decorator
def decorator_talk(talk):

  def child_talk():
    print("inner hello")
    talk()

  return child_talk()


def talk():
  print("this is talk function")

decorator_talk(talk)

"""
inner hello
this is talk function"""
```

DECORATOR

```python
#Decorator
def decorator_talk(talk):

  def child_talk():
    print("inner hello")
    talk()

  return child_talk


def talk():
  print("this is talk function")

func = decorator_talk(talk)
func()

"""
inner hello
this is talk function"""
```

DECORATOR

```python
#Decorator
def decorator_talk(talk):

  def child_talk():
    print("inner hello")
    talk()

  return child_talk

@decorator_talk
def talk():
  print("this is talk function")

talk()
"""
inner hello
this is talk function"""
```

DECORATOR

```python
#Data class
from dataclasses import dataclass
@dataclass
class human:
  name: str
  age: int
  gender: str

ali = human("ali", 20, "M")
```

DECORATOR

```python
#Data class
from dataclasses import dataclass
@dataclass
class human:
  name: str
  age: int
  gender: str

ali = human("ali", 20, "M")
print(ali) #human(name='ali', age=20, gender='M')
```

DECORATOR

```python
from dataclasses import dataclass

@dataclass
class Human:
    name: str
    age: int
    gender: str

    def talk(self):
        print(f"{self.name} is talking.")

ali = Human("Ali", 20, "M")
ali.talk() #Ali is talking.
```

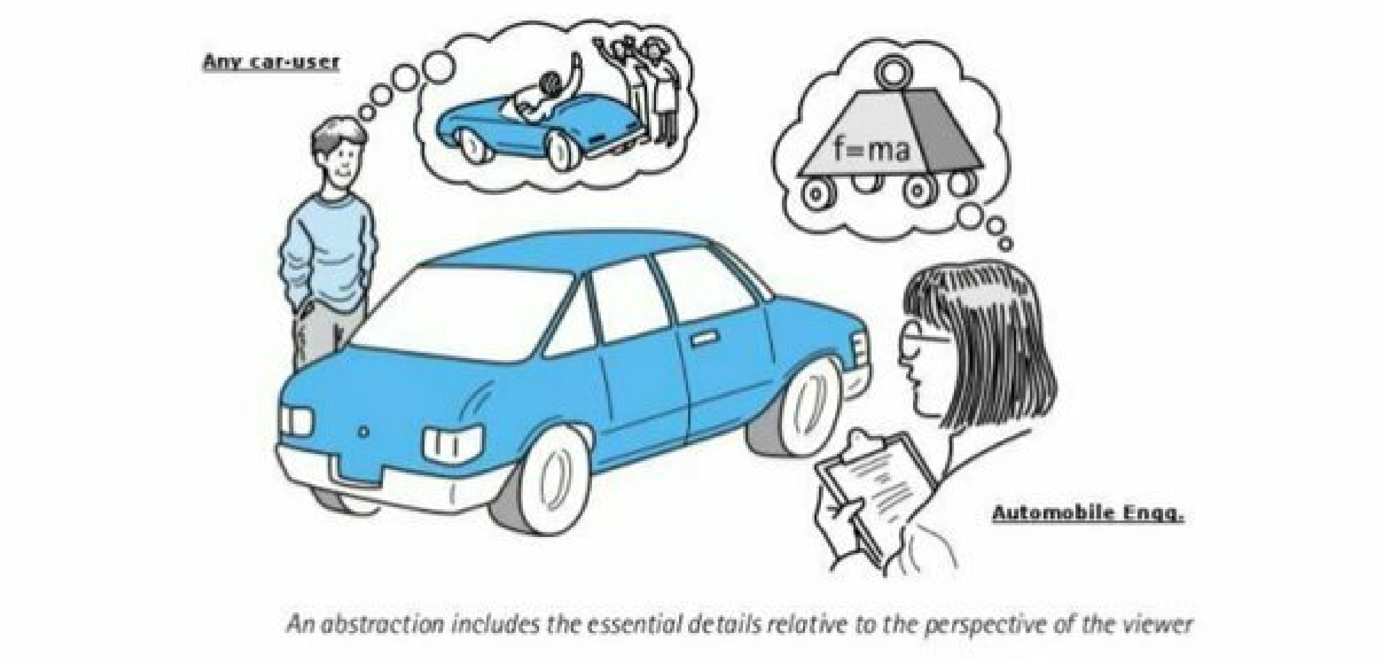
STATIC METHOD

```python
class User:
    users = []

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        self.followers = []
        self.followings = []
        User.users.append(self.email)

    @staticmethod
    def list_users(n):
        return User.users[:n]


maryam = User(name="Maryam", password="secret", email="maryam@example.com")
ali = User(name="ali", password="secret", email="ali@example.com")
print(User.list_users(2)) #['maryam@example.com', 'ali@example.com']

```

CLASS METHOD

```python
from hashlib import md5

class User:
    users = []

    def __init__(self, name, password, email):
        self.name = name
        self.password = password
        self.email = email
        self.followers = []
        self.followings = []
        User.users.append(self.email)

    @classmethod
    def hash(cls, name, password, email):
        hashed_password = md5(password.encode()).hexdigest()
        return cls(name, hashed_password, email)

    def show_pass(self):
        print(self.password)

maryam = User.hash(name="Maryam", password="secret", email="maryam@example.com")
print(maryam.password)
```

ABSTRACTION

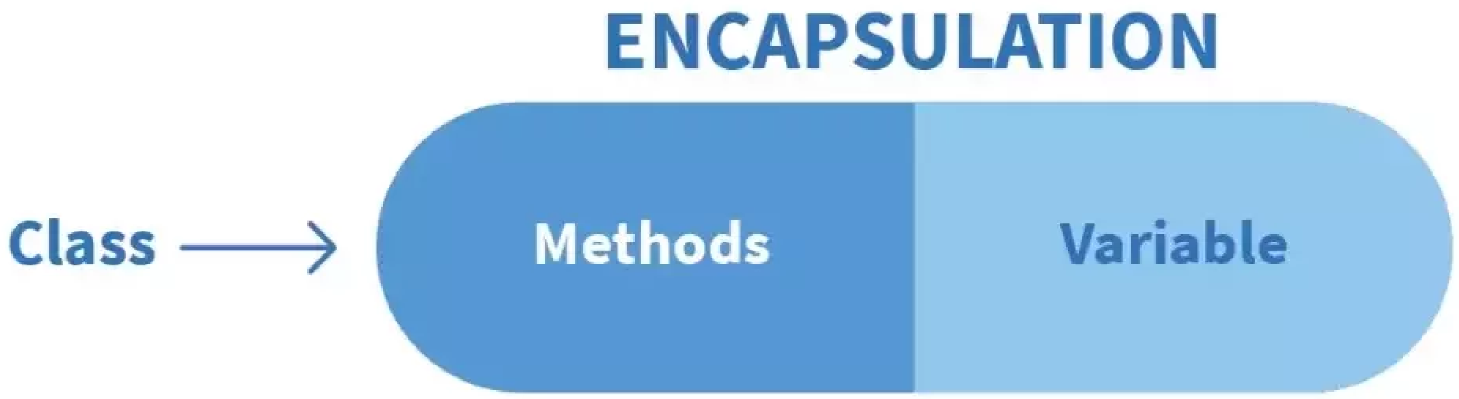


An abstraction includes the essential details relative to the perspective of the viewer

ABSTRACT METHOD

```python
from abc import ABC, abstractmethod

class human(ABC):

    @abstractmethod
    def walk(self):
        pass


class employee(human):
    def walk(self):
        print("walking")



maryam = employee()
maryam.walk()
```

ENCAPSULATION



ENCAPSULATION

```python
#public
class User:
    users = []
    all_followings = {}

    def __init__(self, name="", password=0, email=""):
        self.name = name
        self.password = password
        self.email = email
        self.followers = []
        self.followings = []
        User.users.append(self)
        User.all_followings[self.name] = []

maryam = User(name="Maryam", password="secret", email="maryam@example.com")
maryam.password = "secret2"
print(maryam.password) #secret2
```

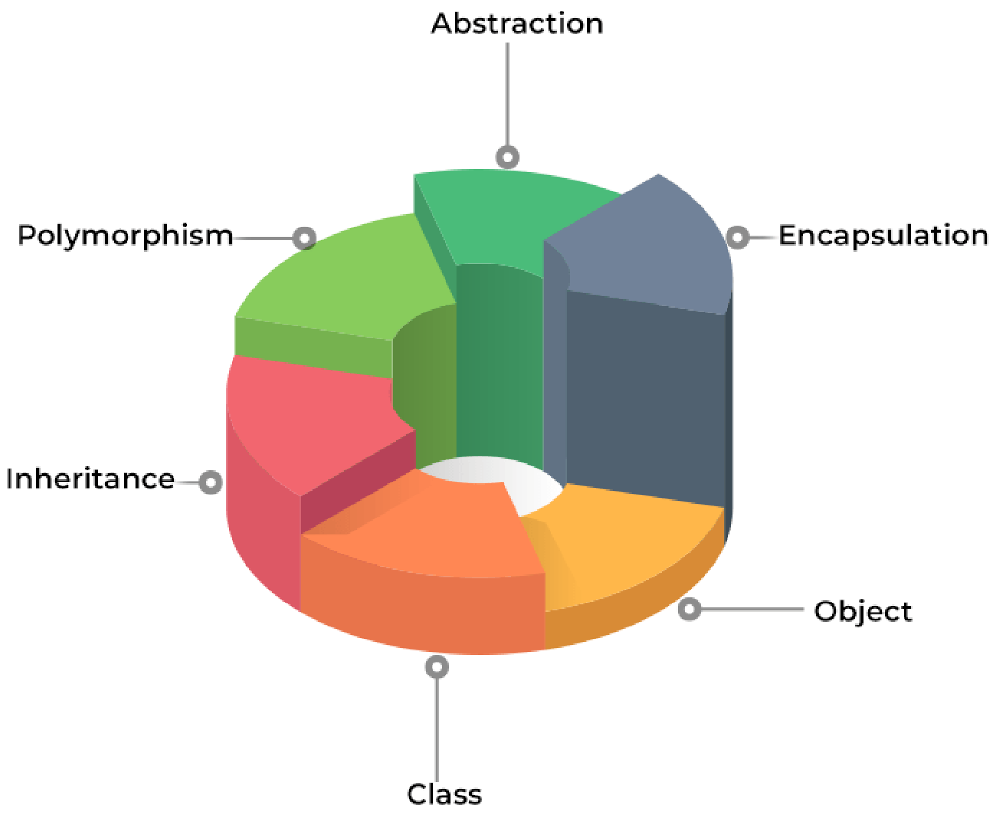
ENCAPSULATION

```python
#private
class User:

    def __init__(self, name, password):
        self.name = name
        self.__password = password


    @property
    def password(self):
        raise AttributeError("its private")

    @password.setter
    def password(self, value):
        #self._passport = value
        raise AttributeError("its private")

maryam = User(name="Maryam",password =123)
maryam.password = "secret2" #AttributeError: its private
print(maryam.password) #AttributeError: its private
```

ENCAPSULATION

```python
#protect
class User:

    def __init__(self, name, password):
        self.name = name
        self._password = password


    @property
    def password(self):
        raise AttributeError("its protected")

    @password.setter
    def password(self, value):
        #self._passport = value
        raise AttributeError("its protected")

maryam = User(name="Maryam",password =123)
maryam.password = "secret2" #AttributeError: its protected
print(maryam.password) #AttributeError: its protected
```

OOP IMPLEMENTATION


Abstraction
Polymorphism
Encapsulation
Inheritance
Object
Class