Tự học Python

Python cơ bản và nâng cao

Tạ Quang Tùng – Pisces Kibo

Mục lục chương trình

19. Giới thiệu về thư viện Pygame

Bài 1: Chương trình đầu tiên

a) **Cách in ra màn hình:**

print(“Hello Python”)
print(‘text’, phép tính)

b) **Chú thích:**

- Chú thích 1 dòng: #Comment
- Chú thích nhiều dòng: ‘’’ text ‘’’ hoặc “”” Text ”””

c) **Toán tử số học:**

“+” Phép cộng
“-” Phép trừ
“*” Phép nhân
“/” Phép chia
“//” Phép chia lấy phần nguyên
“%” Phép chia lấy phần dư
“**” Lũy thừa

d) **Toán tử gán:**

“=” gán giá trị của vế phải cho vế trái
(a=b)
“+=” tăng vế trái bằng một phần vế phải sau đó gán giá trị cho vế trái
(a+=b) → (a=a+b)
“-=” giảm vế trái bằng một phần vế phải sau đó gán giá trị cho vế trái
(a-=b) → (a=a-b)
“*=” nhân giá trị của vế trái với vế phải sau đó gán kết quả cho vế trái
(a*=b) → (a=a*b)
“/=” chia giá trị của vế trái cho vế phải sau đó gán kết quả cho vế trái
(a/=b) → (a=a/b)
“%=” chia giá trị của vế trái cho vế phải sau đó gán phần dư cho vế trái
(a%=b) → (a=a%b)
“//=” phép chia lấy phần nguyên
(a//=b) → (a=a//b)
“**=” lấy vế trái lũy thừa với bậc là vế phải sau đó gán kết quả cho vế trái
(a**=b) → (a=a**b)

e) **Boolean và Toán tử Logic:**

- Giá trị đúng (True), giá trị sai (False)
- “and” phép tính logic “và”
- “or” phép tính logic “hoặc”
- “<”, “<=”, “>”, “>=” lần lượt là các phép so sánh “nhỏ hơn”, “nhỏ hơn hoặc bằng”, “lớn hơn”, “lớn hơn hoặc bằng”
- “==” là phép so sánh bằng nhau
- “!=” là phép so sánh khác
- Mọi số integer khác 0 thì sẽ luôn True

f) String ESCAPE CH:

- Là các ký tự thoát đặc biệt
- Để hiện thị được trong chuỗi thì chèn thêm ký tự backslash \ trước đó

Name	Symbol	Nội dung
Alert	**\a**	ký tự chuông BEL trong chuỗi
Backspace	**\b**	ký tự backspace BS trong chuỗi
Formfeed	**\f**	ký tự Form feed FF trong chuỗi
Newline	**\n**	ký tự xuống dòng trong chuỗi
Carriage return	**\r**	ASCII CF
Horizonal tab	**\t**	ký tự TAB
Vertical tab	**\v**	ASCII VT
Single quote	**\'**	dấu nháy đơn (')
Double quote	**\"**	dấu nháy kép (")
Backslash	****	dấu gạch chéo ngược (\)
Question mark	**\?**	in ra dấu hỏi chấm

Bài 2: Biến, kiểu dữ liệu, ép kiểu dữ liệu

a) Biến:

```
#Khai báo biến a và gán giá trị cho a:
    a = 3
#Khai báo biến b và gán giá trị cho b:
    b = 6
#Tổng của a và b:
    print('a + b =', a+b)
```

b) Kiểu dữ liệu:

- int(): kiểu dữ liệu lưu trữ các số nguyên (1,2,3,4,...)
- float(): kiểu dữ liệu lưu trữ các biến kiểu số thực (1.45, 2.3333, …)
- complex(a,b): kiểu số phức x = a+bi (a,b)
- bool(): kiểu dữ liệu lưu trữ giá trị lý luận (True/False)
- str(): kiểu dữ liệu lưu trữ các xâu ký tự
- type(): kiểm tra kiểu dữ liệu của biến:

```
    name = "PiscesKibo"
    birth = 18
    pi = 3.14
    toantin = True
    x = a + bj
print(type(name))          #string
print(type(birth))         #int
print(type(pi))            #float
print(type(toantin))       #bool
print(x.real, x.imag)      #complex
    → Thực: a, Ảo: b
```

c) Nối chuỗi ký tự:

```
message = "Hello" + " " + "ToanTin"
print(message)
→ Hello ToanTin
```

d) Lưu ý:

- ***Chuỗi ký tự String có thể là 'chuỗi', "chuỗi", """chuỗi"""***
- ***Chỉ có thể nối 2 chuỗi với nhau chứ không thể nối một chuỗi với 1 số***
- ***Để nối một chuỗi với 1 số thì cần đưa số về dạng chuỗi bằng hàm str()***

```
    age = 19
    print("Age: " + str(age))
➔ Age: 19          #string
```

Bài 3: Đầu vào và Toán cơ bản

a) **Đầu vào:**

```
#Nhập dữ liệu từ bàn phím
    name = input()
    print("Hello" + name)
```

Lưu ý:

- ***khi nhập dữ liệu cho một biến từ bàn phím thì kiểu dữ liệu luôn là str***
- ***biến kiểu str thì không tính toán được, nên chuyển dữ liệu về kiểu int()***

```
        age = int(input())
        age = age + 10
        print(age)
```

b) **Toán tử Membership:**

```
    print("Code" in "Codelearn") → True
    print("Py" not in "Python") → False
```

*tương tự: and, or, not, is, ...

c) **Độ ưu tiên Toán Tử:**

Thứ tự ưu tiên	Toán tử	Miêu tả
1	**	Toán tử mũ
2	* / % //	Phép nhân, chia, lấy phần dư, lấy phần nguyên
3	+ -	Toán tử cộng, trừ
4	<= < > >=	Các toán tử so sánh
5	<> == !=	Các toán tử so sánh
6	= %= /= //= -= += **=	Các toán tử gán
7	is, is not	Các toán tử so sánh
8	not, or, and	Các toán tử Logic

Bài 4: Câu lệnh điều kiện IF – ELIF - ELSE

a) Cách 1:

```
if condition:
        #Nếu condition đúng thì khối lệnh này sẽ được thực thi
elif condition:
        #Nếu condition đúng thì khối lệnh này sẽ được thực thi
elif condition:
        ……
else:
        #Nếu condition đúng thì khối lệnh này sẽ được thực thi
```

b) Cách 2:

```
if condition 1:
        #Nếu condition đúng thì khối lệnh này sẽ được thực thi

        if condition 2:
                #Nếu condition đúng thì khối lệnh này sẽ được thực thi
                ……
        ……
```

→ Câu lệnh if lồng nhau Nested-if

c) Cách 3:

```
#Cond = condition đúng thì khối lệnh này sẽ được thực thi

        Cond1 if condition else Cond2
```

d) Ví dụ:

```
        a = 5
        b = 7
#Cách 1:
        if a != b:
            c = 113
        else:
            c = 333
        print(c)
#Cách 2:
        c = 113 if a != b else 115
        print(c)
```

e) Câu lệnh bắt lỗi Try – Except – Finally và Raise Exception:

+ Hỗ trợ try … catch để băt lỗi Runtime, giúp báo rõ loại lỗi chương trình đang gặp vấn đề và tiếp tục hoạt động khi gặp lỗi

vd:

```
try:
        x = 3
        y = 0
        z = x/y
except:
        #Catch
finally:
        # Luôn luôn chạy vào đây với mọi Exception
print("This program has encountered a problem")
```

+ Exception đầy đủ (loại 1):

try:

Thực thi câu lệnh

except <nameError>:

Lỗi đã biết trước

except Exception as error:

Chưa biết được lỗi – Ném ra exception

except:

Ngoại lệ Exception

finally:

Luôn luôn được chạy

```
try:
    Thực hiện các câu lệnh chính

# Lỗi biết trước cụ thể luôn để phía trên cùng
except TypeError:
    # Thực hiện nếu sinh ra lỗi này
except ZeroDivisionError:
    # Thực hiện nếu sinh ra lỗi này
```

```
# Lỗi không biết trước để cuối cùng
except Exception as error:
     # Đưa ra được lỗi cụ thể
     print(f"Lỗi: {error}")

# Khối lệnh finally luôn chạy kể cả khối lệnh try có xảy ra exception hay không?
finaly:
     # Luôn luôn in ra kết quả
```

+ Exeption đầy đủ (loại 2) → raise Exception(“Name Error”)

Bài 5: Vòng lặp While - For

a) So sánh vòng lặp While và For:

Vòng lặp While	Vòng lặp For
+ Vòng lặp while dùng để lặp lại một hành động cho tới khi điều kiện lặp không còn thỏa mãn nữa. Cú pháp vòng lặp while: while <điều kiện>: + Khối lệnh này sẽ được thực thi nếu điều kiện còn đúng + Cấu trúc While có số bước lặp không xác định (vẫn luôn lặp nếu True) vd1: Hiển thị ra màn hình các số từ 1 đến 5: i = 1 while i <=5: print(i) i +=1 vd2: Tính tổng các số từ 1 đến n: n = int(input("Nhập n: ")) i = 1 tong = 0 while i <= n: tong +=i i +=1 print(tong)	+ Vòng lặp for dùng để lặp qua một tập hợp cho trước. Cú pháp vòng lặp for: for <biến> in <gt tham chiếu>: + Khối lệnh thực thi vòng lặp thường được sử dụng với hàm range() range(<điểm đầu>,<điểm cuối>) range(begin,end,step) + Cấu trúc For có số bước lặp xác định vd1: Hiển thị ra màn hình các số từ 1 tới 4: for i in range(1,4): print(i) vd2: In ra màn hình các chữ cái trong chuỗi: a = "ToanTin" for i in a: print(i)

b) Vòng lặp nâng cao While/else và For/else:

While/else	For/else
while condition: while-block else: #Kết quả của while nếu True else-block vd: count = sum = 0 print("Nhập 5 số >= 0 để tính trung bình")	for expression: for-block else: #Kết quả của for nếu True else-block vd: a = int(input("Nhập số nguyên: ")) s = 0

```
while count < 5:
    val = int(input("Nhập giá trị: "))
    if val < 0:
        print("Nhập sai quy tắc")
        break
    sum += val
    count += 1
else:
    print('Trung bình =',sum/count)
```

```
for n in range(5,10):
    if 4 % a == 1:
        print("Ngừng for")
        break
    s += n
else:
    print('S =',s)
```

c) Vòng lặp For qua Pythonlist:

```
#Ban đầu s ở dạng list và Tối ưu hóa vòng lặp
s = [ "<result>" for <biến> in <gt tham chiếu> if <điều kiện>]
print(s)
```

d) Vòng lặp vĩnh cửu While True:

```
While True:
    #Các câu lệnh phía trong được thực thi vô tận
    if "điều kiện":
        #Nếu đúng thì thực thi và thoát ra khỏi vòng lặp
        Break
```

e) Vòng lặp vô hạn với thư viện itertools:

```
import itertools
for n in itertools.count(start = 1):
    #Các câu lệnh phía trong được thực thi vô tận
    if <điều kiện>:
        #Nếu đúng thì thực thi và thoát ra khỏi vòng lặp
        break
```

f) Vòng lặp lồng nhau: (Nested-Loop)

Nested-For	Nested-While
for <biến 1> in <gt tham chiếu 1>: for <biến 2> in <gt tham chiếu 2>:	while <điều kiện 1>: while <điều kiện 2>: ....
Nested-Loop (While – For lồng nhau)	
for <biến 1> in <gt tham chiếu 1>: while <điều kiện 1>:	while <điều kiện 1>: for <biến 1> in <gt tham chiếu 1>: ....

Bài 6: Các câu lệnh đặc biệt (P – B – C)

a) Câu lệnh Pass:

+ Biểu thức "pass" thay thế cho chỗ trống bị lỗi, bỏ qua đoạn đấy

```
if condition:
        pass
else:
        #Nếu đúng thì khối lệnh thực thi
```

b) Câu lệnh Break:

- Thoát khỏi vòng lặp chứa nó trực tiếp khi đạt được mức yêu cầu nào đó (vòng lặp vĩnh cửu)
- Gặp lệnh break, chương trình sẽ không thực hiện bất cứ lệnh nào dưới nó mà thoát ra khỏi vòng lặp luôn

Vd:

```
n = int(input("Nhập n = "))
s = 0
for x in range(1,n+1,1):
    s += x
    if s >= 15:
        break
print('S =',s)
```

c) Câu lệnh Continue:

- Nhảy sớm tới lần lặp kế tiếp, các lệnh bên dưới continue không thực thi
- Gặp countinue chỉ dừng lần lặp hiện tại đang dang dở để chuyển qua lần lặp tiếp theo

Vd:

```
n = 15
s = 0
for x in range(1,n+1,2):
    if x == 3 or x == 11:
        continue
    s += x
print('S =',s)
```

Bài 7: Mảng và liệt kê (Mảng 1 chiều)

a) Khái quát về LIST:

- list là một kiểu dữ liệu cho phép lưu trữ nhiều kiểu dữ liệu khác.
- Để khởi tạo một list thì sử dụng cặp dấu [ ]
  - Khai báo rỗng: lst = []
  - Khai báo list có giá trị: lst = [a,b,c,....]
  - Khai báo list có a phần tử mặc định là x: lst = [x]*a #Kích cỡ list là a
  - Có lst[i] với 0<i<=n là chỉ số index của list

VD: Tạo List

vd1: Tạo list:

```
list1 = [1, 2, 3]
list2 = [“Quân”, “Tùng”, “Thành”]
ist3 = [7, 3, 5, “ToanTin”]
```

→ OUTPUT:

```
[1, 2, 3]
['Quân', 'Tùng', 'Thành']
[7, 3.5, 'ToanTin']
```

vd2: Truy xuất các phần tử trong list dùng toán tử []:

```
names = [“Quân”, “Tùng”, “Thành”]
print(name[0])
print(name[1])
print(name[2])
```

→ OUTPUT

```
Quân
Tùng
Thành
```

#Có name[i] với 0<=i<n là index của LIST

b) Các hàm phương thức xử lý với List:

- **Hàm len:**
  - Trả về số phần tử có trong list
  - Ví dụ: trả về 3 phần tử trong lst

```
lst = [2, 3, 1]
print(len(lst))
```

KQ: 3

- **Hàm count:**
  - Đếm số lần xuất hiện của một thành phần trong list
  - Ví dụ: số lần xuất hiện:

```
lst = [6, 2, 3, 8, 2]
print(lst.count(2))
```

KQ: 2

- **Hàm max, min:**
  - Đây là hai hàm dùng để trả về phần tử MAX và MIN trong list
  - Ví dụ:

```
lst = [2, 3, 1]
print(max(lst))
print(min(lst))
```

KQ: 3 và 1

- **Hàm sort:**
  - Hàm này dùng để sắp xếp các phần tử trong list theo thứ tự nhất định
  - Ví dụ: #Có làm thay đổi ND list ban đầu

```
lst = [4, 5, 3, 7, 6, 1]
# Sắp xếp các phần tử theo thứ tự tăng dần
lst.sort()
print(lst)
# Sắp xếp các phần tử theo thứ tự giảm dần
lst.sort(reverse=True)
print(lst)
```

KQ:

```
[1, 3, 4, 5, 6, 7]
[7, 6, 5, 4, 3, 1]
```

  - Hàm sorted có thể đứng độc lập nhưng chức năng như hàm sort
    vd: lst = sorted(lst) #Không làm thay đổi ND list ban đầu

- **Hàm remove:**
  - Hàm này để xóa một phần tử element khỏi list (first)
  - Cấu trúc: ***lst.remove(element)***
  - Ví dụ:

```
lst = ['A', 'B', 'C']
lst.remove('A')
print(lst)
```

KQ: [‘B’, ‘C’]

hoặc del lst[0]

  - Hàm del lst[a:b] xóa từ phần tử thứ a tới b-1 của list

- **Hàm pop:**
  - Dùng để xóa một phần tử với chỉ số cho trước trong list

- Cấu trúc: ***lst.pop(index)***
- Ví dụ:

```
lst = ['A', 'B', 'C']
# Xóa phần tử thứ 2 khỏi list
lst.pop(1)
print(lst)
```

KQ: [‘A’, ‘C’]

- **Hàm append:**
  - Để thêm một phần tử vào cuối của list
  - Cấu trúc: ***list1.append(element)***
  - Ví dụ

```
lst = []
lst.append(4)
lst.append(3)
lst.append(6)
print(lst)
```

KQ: [4, 3, 6]

- **Hàm insert:**
  - Hàm dùng để thêm một phần tử vào một vị trí bất kỳ trong list
  - Cấu trúc: ***insert(index, newElement)***
  - Ví dụ:

```
lst = ['a', 'e', 'i', 'u']
lst.insert(3, 'o')
print(lst)
```

KQ: [‘a’, ‘e’, ‘i’, ‘o’, ‘u’]

- **Hàm extend:**
  - Hàm nối hai list lại với nhau
  - Cấu trúc: ***list1.extend(list2)***
  - Ví dụ:

```
list1 = [1, 6, 7, 9]
list2 = [2, 4, 3, 5]
list1.extend(list2)
```

KQ: [1, 6, 7, 9, 2, 4, 4, 5]

- **Hàm clear:**
  - Hàm dùng để xóa hết các phần tử bên trong list
  - Ví dụ:

```
lst = [1, 2, 3]
lst.clear()
print(lst)
```

KQ: [ ]

- **Hàm reverse:**
  - Đây là hàm đảo ngược list
  - Ví dụ:

```
lst = [4, 5, 3, 7, 6, 1]
lst.reverse()
print(lst)
```

KQ: [1, 6, 7, 3, 5, 4]

  - Hàm reversed giống reverse nhưng có thể đứng độc lập

```
lst = [6,0,2,100,20]
lst2 = reversed(lst)
for item in lst2:
    print(item)
```

- **Hàm index và rindex:**
  - index(a): chỉ vị trí của phần tử a đầu tiên (a ở vị trí nào)
  - rindex(a): trả về vị trí cuối cùng của phần tử a

- **Hàm del:**
  - Xóa sạch list đó khiến nó không tồn tại

- **Hàm replace:**
  - a.replace("nội dung muốn đổi", "nội dung mới")
    (với a là string ban đầu)

- **Hàm enumerate:**
  - Có sự kết hợp chặt chẽ giữa vòng lặp Pythonist
  - Liệt kê các phần tử của list với chỉ số tương ứng (2 cặp giá trị cùng chạy vòng lặp)

Vd:

```
#Cách 1:
for i, j in enumerate(a):    #liệt kê các phần tử của list với chỉ số t/ứg
  lst.append(j + a2[i])
print(lst)

#Cách 2:
ln = [j + a2[i] for i, j in enumerate(a)]      #enumerate in Pythonlist
print(ln)

#Cách 3:
```

```
for i, x in enumerate(a):
  print(i, x)
print(a)
```

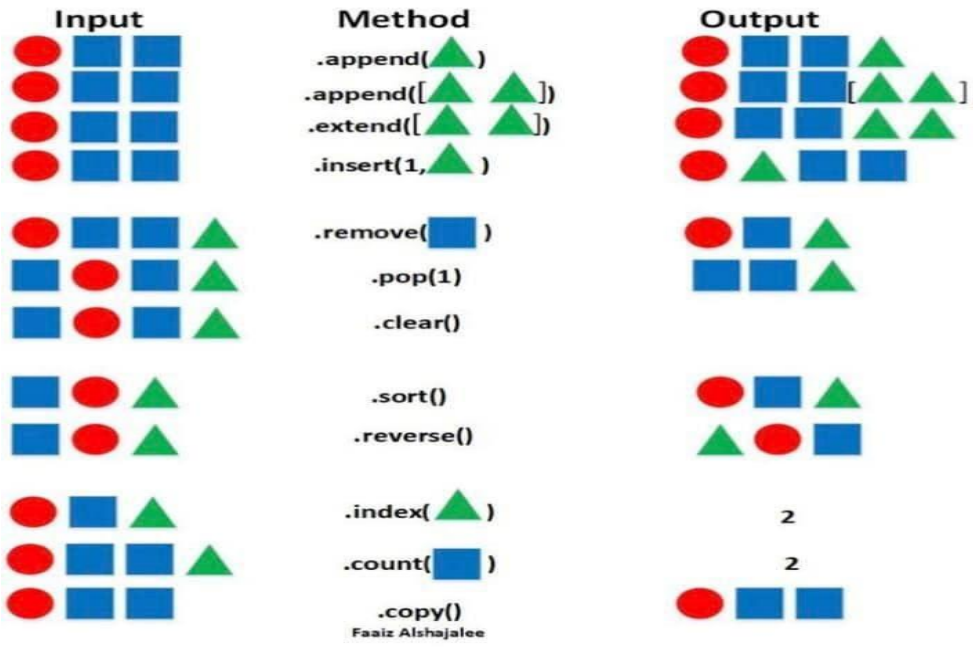
- **Hàm zip:**
  - zip lại phần tử A[i] theo phần tử B[j] lại với nhau
  - Cơ bản giống hàm enumerate()

vd:

```
#Cách 1:
for x, y in zip(a, a2):
  print(x,y)
print(a)
print(a2)

#Cách 2: (tổng x và y)
print([x + y for x, y in zip(a, a2)])
```

c) Đơn giản hóa 1 vài phương thức với List:



d) Cách duyệt list:

+ Duyệt theo collection
+ Duyệt theo list

lst = [5,7,2,9,6,3,10,17,16]	
#Duyệt theo collection	#Duyệt theo chỉ số index

for x in lst: print(x,end = '\t')	*+ Theo thứ tự list* for i in range(len(lst)): x = lst[i] print(x,end = '\t') print() *+ Ngược chiều list* for i in range(len(lst)-1,-1,-1): x = lst[i] print(x, end = '\t')

e) Cách trích lọc list – Slicing:

+ Slicing dùng để trích lọc list

list [begin : end : step]

+ Có thể lấy phần tử cuối cùng bằng cách list[-1]

Bài 8: Mảng đa chiều dữ liệu Turple

a) Mảng nhiều chiều Array:

<Tên> = [<Giá trị>]

vd: Lst = [1,2,3, [“Tùng”, “Quang”, “Tạ”] , [“Toán tin”, “MIM”, “HUS”] , [6,3,2003]]
vd: Lst = [[1,2,3],
[4,5,6],
[7,8,9]]

=> print(Lst) #Dạng Metrix

b) Cách gọi phần tử mảng 2D:

<tên list>[i] [e]

Trong đó <tên list> là tên list chứa giá trị cần:
i là vị trí của list con trong list lớn
e là vị trí phần tử cần lấy giá trị trong list con

vd: Lst[0] = 1
Lst[3] [0] = ‘Tùng’
Lst[4] = “Toán tin”, “MIM”, “HUS”

c) Kiểu dữ liệu Turple:

tup = (pt1,pt2,...)

vd: tup = (1,2,3)
tup2 = (“Tạ”, “Quang”,”Tùng”)
tup3 = (True, “MIM”, 63)
tup4 = (1,2,)

- Cách thêm dữ liệu vào trong tuple: ***tuple += (giá trị,)***
- Đặc điểm:
  - Lưu trữ các phần tử theo thứ tự đã cho và không thay đổi được
  - Cho phép dữ liệu trùng lặp và các kiểu dữ liệu khác nhau

d) Hàm phương thức trong tup:

- Nếu trong list toàn là chuỗi có thể thay i là item

```
lst = ["pisces", "kibo", "putin"]
for item in lst:
    print(item,end=' ')
```

- Giống như list thì turple có thể làm được
  - Row dòng, Column cột
    - Lst = [[a]*column]*row

- Có thể lấy list + list

- Cách gộp kiểu dữ liệu Tuple() là:
  - tup = () *#Tạo mảng ban đầu*
  - tup += ()

- Cách lấy giá trị phần tử:
  - Truy xuất phần tử giống list → ***tuple1[index]***
  - Lấy giá trị từng phần tử:

```
tuple1 = (element1, element2, element3, ...)
 (x1, y1, z1, ...) tuple1
→ Mọi biến đều được lấy giá trị từ tuple
```

  - Lấy giá trị một vài phần tử:

```
tuple1 = (element1, element2, element3, ...)
(x1, y1, *_) = tuple1
→ x1 và y1 được gán giá trị, còn lại thì không
```

  - Lấy phần tử ở vị trí khác nhau:

```
tuple1 = (element1, element2, element3, ...)
(x1, _, x3, x4, *_) = tuple1
→ x1, x3, x4 được gán giá trị, còn lại thì không
```

    → _ để bỏ trống 1 giá trị
    → *_ để bỏ trống tất cả các giá trị đằng sau (luôn để cuối cùng)

e) Kiểu dữ liệu Dictionary và Set:

- Chú thích: set và dictionary cũng giống như list hay tup

- Kiểu dữ liệu DICTIONARY:
  - Khởi tạo: dict = {}
  - Kiểu dạng cơ bản: dict = { key1 : value1, key2 : value2, …..}
  - Kiểu dạng lồng:

```
dict = { key1 : value1, key2 : { key21 : value21, key22 : value22, ...}, key3: value3, ... }
```

  - Kiểu đối tượng:

```
dictionaryObject = dict(
        key1 = value1,
        key2 = value2,
        ...
)
```

- Các phần tử phải có key duy nhất dạng số hoặc dạng chuỗi
- Nếu khai báo key thì không thể đổi tên
- Key có phân biệt HOA và thường

- Lấy value tương ứng khi có key → ***dictionary[key] = value***
- Cách duyệt dictionary:

items là cả keys và values

```
for key, val in dict.items()        => Duyệt cả key và value
for item in dict.items()            => Lấy cặp key và value
for i in dict.keys()                => Chỉ duyệt các key
for I in dict.values()              => Chỉ duyệt các value
```

- Các phương thức Dictionary:
  - Thêm item bằng add() → dictionary1[newKey] = newValue
  - Thêm dictionary2 bằng update() → dictionary1.update(dictionary2)
  - Xóa dictionary → del dictionary1[key]
  - Xóa key → dictionary1.pop(key)
  - Xóa sạch dict tồn tại → dictionary1.clear()
  - Phương thức lấy toàn bộ keys → my_dict.keys()
  - Phương thức lấy toàn bộ values → my_dict.values()
  - Độ dài của Dict → ***len(dict)***

- Kiểu dữ liệu SET:
  - Hàm set không có dạng list hay turple nên không có cùng tính chất
  - Mặc định xóa bỏ phần tử trùng lặp và tự động sắp xếp
  - Để duyệt SET thì phải dùng vòng lặp
  - Độ dài của SET là: ***len(set)***
  - Cách chuyển set về list là ép kiểu dữ liệu list vào set:

```
ln = set(lst)          #ln thành kiểu set
mang = list(ln)        #ln chuyển lại về list
```

  - Các loại toán tử trong SET:

Toán tử	Mô tả	Ví dụ
-	Trả về là một Set gồm các phần tử chỉ tồn tại trong Set1 mà không tồn tại trong Set2 (tồn tại set1 nhưng không tồn tại trong set2)	> {1, 2, 3} - {2, 3} : {1} > {1, 2, 3} - {4} : {1, 2, 3} > {1, 2, 3} - {1, 2, 3} : set() > {1, 2, 3} - {1, 2, 3, 4} : set()
&	Trả về là một Set chứa các phần tử vừa tồn tại trong Set1 vừa tồn tại trong Set2 (lấy phần chung)	> {1, 2, 3} & {4} : set() > {1, 2, 3} & {1, 2, 3} : {1, 2, 3} > {1, 2, 3} & {1, 2, 3, 4} : {1, 2, 3}
|	Trả về là một Set chứa tất cả các phần tử tồn tại trong hai Set (gộp các Set với nhau)	> {1, 2, 3} | {2, 3} : {1, 2, 3} > {1, 2, 3} | {4} : {1, 2, 3, 4} > {1, 2, 3} | {1, 2, 3} : {1, 2, 3} > {1, 2, 3} | {1, 2, 3, 4} : {1, 2, 3, 4}

- Các phương thức trong SET:

Phương thức	Mô tả
set.add(element)	Thêm một phần tử vào SET
set.remove(element)	- Loại bỏ một phần tử ra khỏi tập hợp - Nếu item không tồn tại sẽ báo lỗi
set.discard(element)	- Loại bỏ một phần tử ra khỏi tập hợp - Nếu item không tồn tại sẽ không báo lỗi
set.pop()	Loại bỏ một phần tử ngẫu nhiên ra khỏi tập hợp
set.clear()	Loại bỏ hết các phần tử ra khỏi tập hợp
set.update(item1, item2, ...)	- Thêm nhiều item vào tập hợp - Nối các set vào nhau, remove duplicate
set1. intersection(set2)	Giống toán tử &
set1. difference(set2)	Giống toán tử -
set1. symmetric_difference(set2)	Giống toán tử ^
set1.union(set2)	Giống toán tử |

f) Collection trong Python:

- Collection list → [ ] → Hàm khởi tạo: list()
- Collection tuple → ( ) → Hàm khởi tạo: tuple()
- Collection set → { } → Hàm khởi tạo: set()
- Collection dictionary → { } → Hàm khởi tạo: dict()
  => Giống ép kiểu dữ liệu

Bài 9: Các hàm toán học cơ bản

a) Hàm round():

+ Hàm làm tròn số round(a,b) → a là số cần làm tròn, b là làm tròn đến mấy chữ số thập phân

- Làm tròn lên 1 số → math.ceil(number)
- Làm tròn xuống 1 số → math.floor(number)

b) Hàm end():

+ Làm output không xuống dòng

vd:

```
for i in range(10):
        print(n, end = ' ')
```

→ 0 1 2 3 4 5 6 7 8 9

c) Hàm exit(): (Đối với vòng lặp vĩnh cửu):

+ Dùng để thoát phần mềm (thoát hẳn chương trình)

vd:

```
while True:
        s = input("Tên bạn: ")
        print(s)
        hoi = input("Tiếp không (c/k) ? : ")
        if hoi == 'k':
                exit()
print("Bye")
```

d) Các import thư viện Toán học (Math):

- import math → dùng 1 hàm toán học
  #Các câu lệnh hàm chứa toán: "math.cthtoan"
- from math import * → dùng nhiều hàm toán học
  #Câu lệnh chứa hàm toán:
  - Sqrt(a) → Căn bậc hai của a
  - Pow(a,b) → Lũy thừa (a^b)
  - Log → log(x) = loge(x) = lnx
  - Log10 → Logarit cơ số 10 của x, log10(x)
  - logax → lnx/lna
  - log(x, a) → log cơ số a của x
  - Exp(x) → tính e^x
  - Degrees(a) → Đổi radian ra độ
  - Radians(a) → Tính radian 180/PI*x
  - Fabs → Tính giá trị tuyệt đối
  - sinx = sin(radians(x))

- cosx = cos(radians(x))
- tanx = tan(radians(x))
- gcd → gcd(val1, val2) #Tìm UCLN

e) Hàm eval():

+ Tối ưu phép tính bằng cách tự tính toán chuỗi các phép toán khó

vd:

```
from math import sin
x = eval("1+2+5+sin(30)")
print(x)
```

+ Gán biến nhiều giá trị cùng 1 dòng

vd:

```
x1,x2 = eval(input("Nhập x1, x2 :"))
print('x1=',x1,'x2',x2)
print('{0}+{1}={2}'.format(x1,x2,x1+x2))
```

f) Hàm time:

+ from time import …

- clock() → khoảng thời gian = end – start

vd:

```
from time import clock
start = clock()
print("Mời bạn nhập 1 giá trị: ")
x = input()
print('Bạn nhập x =',x)
end = clock()
duration = end - start
print('duration =',duration)
```

- time() → giống clock()

vd:

```
import time

t1 = time.time()

time.sleep(5)
time.sleep(5)

t2 = time.time()
print(f"Time speed: {t2 - t1}")
```

- sleep() → tạm dừng chương trình chạy trong 1 khoảng thời gian

vd:

```
from time import sleep
for count in range(10,-1,-1):
    print(count)
sleep(1)
```

g) Hàm random:

- randrange(x,y) —> Lấy số bất kỳ >=x và <y
- randint(x,y) —> Lấy số bất kỳ >= x và <= y
- Câu lệnh:

```
from random import randrange
    x = randrange(x,y)
print(x)
```

h) Thư viện Module DATETIME:

- Module datetime để xử lý ngày giờ cùng các mốc thời gian
  - Import thư viện chung → ***import datetime***
  - Import thư viện cụ thể → ***from datetime import ****

- Các hàm xử lý ngày giờ Datetime chung:

Câu lệnh	Ý nghĩa
Today = datetime.date.today()	Ngày, tháng, năm hiện tại (year – moth – day)
Today = datetime.datetime.today()	Thời gian hiện tại
today = datetime.date(year, moth, day)	Ngày, tháng, năm tùy ý
today.year	Năm hiện tại
today.moth	Tháng hiện tại
today.day	Ngày hiện tại
Date(string)	Chuyển về dạng datetime phù hợp
timeName = time(hour, minute, second)	Thời gian tùy chọn
timeName.hour	Giờ
timeName.minute	Phút
Time.microsecond	Giây
datetime(year, month, day, hour, minute, second, ...)	Đầy đủ thời gian
t1 = timedelta(weeks, days, hours, ...) t2 = timedelta(weeks, days, hours, ...) → t3 = t1 – t2	Thực hiện phép tính với thời gian chênh lệch

- Định dạng ngày tháng bằng strftime() → Format time
  - nameTime.strftime("stringTime")
    - %Y - year [0001,..., 2018, 2019,..., 9999]
    - %m - month [01, 02, ..., 11, 12]
    - %d - day [01, 02, ..., 30, 31]
    - %H - hour [00, 01, ..., 22, 23
    - %M - minute [00, 01, ..., 58, 59]
    - %S - second [00, 01, ..., 58, 59]

  - Chuyển String sang Datetime:

    - datetime.strptime(“dateString”, <định dạng format>)

- Tính tổng số giây → *timeVariable.total_seconds()*
- Đổi số giây về thời gian Date → *date.fromtimestamp(secondNumber)*

- Hiển thị múi giờ Timezone với module pytz
  → pytz.timezone(“NationalLanguage”)

Bài 10: Tổng quan hàm Format

a) Hàm format:

+ print('{0} {1}'.format(a,b)) → Tức a thay cho vị trí {0} và b thay cho vị trí {1} —> để dành chỗ

vd:

'{0} {1}'.format(7, 10**7) → '7 10000000'
'a{0}b{1}c{0}d'.format('x', 'y') → 'axbycxd'
'{0}/{1} = {2}'.fomat(a,b,a/b)

+ print('{0 :> 3} {1 :> 10}'.format(a, b) → Tức a cách đầu dòng 3 dấu cách và b cách đầu dòng 10 dấu cách

b) Format String dữ liệu:

- print(f"The solution is: x = {x}") với x có sẵn ở phía trên
  vd: (giá trị kết quả của x là {x})

```
x = -c/b
print(f"The solution is: x = {x}")    #Format String
```

c) Hàm Substring:

- Hàm trích lọc chuỗi Slicing: (dùng cho cả String và List)

  list [begin : end : step]

  - list: là danh sách
  - begin: vị trí bắt đầu cắt
  - end: vị trí cuối cùng cắt
  - step: bước nhảy

  - x[a:] → bỏ a ký tự đầu của chuỗi, chỉ lấy phần còn lại
  - x[:a] → chỉ lấy a ký tự đầu, còn lại bỏ đi
  - x[:-a] → bỏ a ký tự cuối, chỉ lấy phần phía trước
  - x[-a:] → lấy a ký tự cuối, còn lại phía trước bỏ đi
  - x[a:-a] → bỏ a ký tự đầu và bỏ a ký tự cuối, còn lại giữa nguyên
  - x[a:b] → lấy từ ký tự a đến ký tự b
  - x[:] → lấy hết tất cả
  - x[::a] → lấy từ đầu tới cuối với bước nhảy là a

```
x = "Hello World!"
print(x[2:])        #"llo World"
print(x[:2])        #"He"
print(x[:-2])       #"Hello Worl"
print(x[-2:])       #"d!"
```

```
print(x[2:-2])        #”llo Worl”
print(x[6:11])         #”World”
```

Bài 11: Xử lý chuỗi qua mảng

a) Hàm split() – Tách chuỗi thành mảng:

- Tách 1 chuỗi thành nhiều chuỗi và các chuỗi con tạo thành 1 list
- Cú pháp:
  <Chuỗi muốn tách>.split(<chuỗi dùng để cắt>,<số chuỗi muốn cắt thêm>)

  *<số chuỗi muốn cắt thêm> có thể lớn hơn số chuỗi muốn tách
  vd1:

a="K 6 6 A 2" b=a.split(" ",4) print(b) → ['K', '6', '6', 'A', '2']	+ Chuỗi dùng để cắt là " " (Khoảng trắng) - mặc định + Số chuỗi muốn cắt thêm là: 4: nếu bạn muốn cắt thành 5 chuỗi thì số chuỗi muốn cắt sẽ phải trừ đi 1

vd2:

a="K 6 6 A 2" b=a.split() print(b) → ['K', '6', '6', 'A', '2']	Nếu không nhập bất cứ tham số bên trong nào của split() thì chương trình vẫn chạy được vì nếu không nhập thì split() sẽ dùng các giá trịmặc định.

b) Một số cách tách chuỗi liền:

Dùng vòng for	Ép kiểu dữ liệu
a="456" b=[] for i in a: b.append(i) print(b) → ['4', '5', '6']	a="456" b=list(a) print(b

c) Các phương thức nhập input() khác:

- **Hàm map():**
  - Nhập nhiều input() cùng 1 dòng
  - Cú pháp:
    <biến 1>,<biến 2>,...= map(<kiểu dữ liệu>,input().split())

    vd:

a,b=map(str,input().split()) print("a=",a,type(a)) print("b=",b,type(b))	#Nhập input 4 5 #Kết quả a= 4 <class 'str'> b= 5 <class 'str'>

- Nhập nhiều giá trị của list mà không nhập trước số lượng: (hàm tách chuỗi)

Nhập nhiều phần tử trên 1 dòng	Nhập nhiều phần tử trên nhiều dòng
s = input().split() print(s)	n=input() lst=[] while n!="stop": lst.append(n) n=input() print(lst)

vd:

s = "sv123;Tạ Quang Tùng;6/3/2003" *arr = s.split(';')* *print(arr)* *for x in arr:* * print(x)* *—> ["sv123", "Tạ Quang Tùng", "6/3/2003"]* *—> sv123* *—> Tạ Quang Tùng* *—> 6/3/2003*	*nếu chuỗi ban đầu nhiều dòng có thể dùng splitlines() *s = """Obama* * hahaha* * ali333"""* *arr = s.splitlines()* *for line in arr:* * print(line,"a→", line.count("a"))*

- **Hàm nối chuỗi join():**
  - Có thể nối chuỗi riêng biệt bằng dấu cộng '+'

s = "sv123;Tạ Quang Tùng;6/3/2003" *arr = s.split(';')* *for x in arr:* * print(x)* *s2 = ","* *s2 = s2.join(arr)* *print(s2)*	*print(x):* *+ sv123* *+ Tạ Quang Tùng* *+ 6/3/2003* *print(s2):* *sv123,Tạ Quang Tùng,6/3/2003*

Bài 12: Các phương thức xử lý chuỗi String

a) Tổng quan về chuỗi:

- Chuỗi là tập ký tự trong nháy đơn hoặc nháy đôi, nếu chuỗi xuống dòng dùng 3 nháy đơn hoặc 3 nháy đôi
- Chuỗi có 1 số hàm quan trọng:

object 。method name (parameter list)

  - object: là tên chuỗi
  - method name: tên phương thức của nó
  - parameter list: là các đối số nếu có

- upper, lower : xử lý in Hoa, in thường
- rjust : căn lề phải
- ljust : căn lề trái
- center : căn giữa
- strip : xóa khoảng trắng dư thừa
- startswith : kiểm tra chuỗi có phải bắt đầu là ký tự ?
- endswith : kiểm tra chuỗi có phải kết thúc là ký tự ?
- count : đếm số lần xuất hiện trong chuỗi
- find : tìm kiếm chuỗi con
- format : định dạng chuỗi
- _len_() : trả về số lượng ký tự trong chuỗi, dùng index để lấy ký tự ra: str[index] #len()

b) Phân tích các hàm xử lý chuỗi:

a) Hàm upper, lower – in HOA/thường:

vd: Hàm upper in HOA: name = “tạ quang tùng” print(name.upper()) —> TẠ QUANG TÙNG	vd: Hàm lower in thường: name = “TẠ QUANG TÙNG” print(name.lower()) —> tạ quang tùng

b) Hàm title – tự động viết hoa chữ cái đầu sau khi đã tối ưu khoảng trắng:

s = “chuỗi”.title()

vd:

```
s = “hello python”
print(s.title())
—> Hello Python
```

c) Hàm căn lề rjust, ljust, center:

- **rjust - căn phải chuỗi**:
  - Nếu truyền 1 đối số → mặc định là khoảng trắng phía trước
  - Nếu có đối số thứ 2 → chèn đối số 2 vào trước

word.rjust(a, “chuỗi”)

a là số ký tự trong chuỗi
nếu word ban đầu không đủ a ký tự thì chèn trước word "chuỗi"
nếu không viết "chuỗi" thì mặc định là khoảng trắng
nếu a < word thì vẫn không thay đổi gì bằng word

word = "ABCD" print(word.rjust(10, '*')) print(word.rjust(3, '*')) print(word.rjust(10))	******ABCD #Nếu word không đủ 10 ký tự, chèn * ABCD #Nếu word có ký tự > 3 ABCD #vì không đối số thứ 2 nên trước word sẽ là khoảng trắng

- **ljust - căn trái chuỗi:**
  - Nếu truyền 1 đối số → mặc định là khoảng trắng phía sau
  - Nếu có 2 đối số → chèn đối số 2 vào sau

word.ljust(a, "chuỗi")
a là số ký tự trong chuỗi
nếu word ban đầu không đủ a ký tự thì chèn sau word "chuỗi"
nếu không viết "chuỗi" thì mặc định là khoảng trắng
nếu a < word thì vẫn không thay đổi gì bằng word

word = "TUVOI" print(word.ljust(2)) print(word.ljust(5)) print(word.ljust(10, '*'))	TUVOI #word có số ký tự > 2 TUVOI #đối số 2 không có nên mặc định khoảng trắng TUVOI***** #word không đủ 10 ký tự thì chèn thêm '*' phía sau cho đủ

- **center - căn giữa chuỗi:**
  - Nếu 1 đối số → đẩy hai khoảng trắng ra hai bên sao cho đủ ký tự
  - Nếu 2 đối số → thay hai khoảng trắng hai bên bằng ký tự mới sau này

word.center(a, "chuỗi")
a là số ký tự trong chuỗi
nếu word ban đầu không đủ a ký tự thì chèn hai bên word "chuỗi"
nếu không viết "chuỗi" thì mặc định hai bên là khoảng trắng
nếu a < word thì vẫn không thay đổi gì bằng word

  - Ưu tiên đặt nhiều hơn sau word (bên phải) nếu bị lẻ

word = "TUVOI" print(word.center(10)) print(word.center(10, '*'))	TUVOI #không có đối số 2 nên mặc định hai bên là khoảng trắng **TUVOI*** #word không đủ 10 ký tự chèn thêm '*'

d) Hàm strip – xóa khoảng trắng dư thừa:

- chỉ xóa được khoảng trắng dư thừa bên trái hoặc bên phải
- trong hàm strip(^^) với ^^ có thể bỏ trống hoặc ký tự khác

vd:

```
s = “  ABCDEFGHIJKLMNOPQRSTUVWXYZ  “
print(s)
print(s._len_())
s = s.strip()
print(s)
print(s._len_())
—>  ABCDEFGHIJKLMNOPQRSTUVWXYZ    #chứa khoảng trắng
—> 34                              #độ dài ban đầu
—>ABCDEFGHIJKLMNOPQRSTUVWXYZ      #sau khi xóa
—> 32                              #độ dài lúc sau
```

e) Hàm startswith và endswith:

- startswith kiểm tra chuỗi có bắt đầu bằng 1 chuỗi con nào đó không
- endswith kiểm tra chuỗi có kết thúc bằng 1 chuỗi con nào đó không

```
s = “#hello Python*”
print(s.startswith(“#”))      #True
print(s.startswith(“$”))      #False
print(s.endswith(“#”))        #False
print(s.endswith(“*”))        #True
```

f) Hàm find và count:

- **Hàm find và rfind:**
  - Hàm find trả về vị trí đầu tiên tìm thấy
  - Hàm rfind trả về vị trí cuối cùng tìm thấy (nếu không thấy sẽ trả -1)

```
s = “hello hello hello”
x1 = s.find(‘o’)
print(x1)                 —> 4
x2 = s.rfind(‘o’)
print(x2)                 —> 16
x3 = s.find(‘x’)
print(x3)                 —> -1
```

- **Hàm count:**
  - Hàm count trả về số lần xuất hiện của chuỗi con trong chuỗi gốc
  - Nếu không tồn tại sẽ trả về 0

```
s = “Obama likes Putin, Putin likes Kim Jong Un”
cl = s.count(“Putin”)
print(cl)                    —> 2
c2 = s.count(“Trump”)
print(c2)                    —> 0
```

- s.count(‘chuỗi’) —> đếm số lần xuất hiện ‘chuỗi’ trong s
- s.count(‘chuỗi’,a) —-> đếm số lần xuất hiện ‘chuỗi’ kể từ vị trí a
- s.count(‘chuỗi’,a,b) —> đếm số lần xh ‘chuỗi’ kể từ a đến b trong s

g) Hàm isdecimal(), isdigit(), isumeric():

- Hàm isdecimal() để kiểm tra chuỗi có là dạng số nguyên không
- Hàm isdigit() để kiểm tra chuỗi có là dạng số thập phân không
- Hàm isnumeric() trả về True nếu một chuỗi dạng Unicode chỉ chứa các ký tự số, nếu không là false (dùng nhiều nhất bao gồm cả isdecimal() và isdigit())

vd:

str = “k66taquangtung”
print(str[i].isumeric()) #Xem ở vị trí i có là dạng số không

```
"""
+-------------+-----------+-------------+---------------------------------+
| isdecimal() | isdigit() | isnumeric() |          Example                |
+-------------+-----------+-------------+---------------------------------+
|    True     |    True   |    True     | "038", "০৩৮", "０３８"          |
|   False     |    True   |    True     | "০৩৮", "0.3.8.", "⓪③⑧"         |
|   False     |   False   |    True     | "⅓¼⅝", "IIIVIII", "⓪③⑤⓪", "壹貳參"  |
|   False     |   False   |   False     | "abc", "38.0", "-38"            |
+-------------+-----------+-------------+---------------------------------+
"""
```

isdecimal()⊆ isdigit()⊆ isnumeric()

h) Hàm min() và max():

- Đây là phương thức tìm ký tự có giá trị lớn nhất và nhỏ nhất:
  - min(string) → Ký tự có giá trị nhỏ nhất trong chuỗi
  - max(string) → Ký tự có giá trị lớn nhất trong chuỗi

Bài 13: Cách xây dựng hàm Function

a) Khái niệm về hàm:

- Khối lệnh thực thi một công việc hoàn chỉnh, được đặt tên và gọi thực thi nhiều lần tại nhiều vị trí trong chương trình
  ➔ Hàm còn được gọi là chương trình con
- Gồm hàm nhiều thư viện có sẵn (from … import …) và hàm do người dùng định nghĩa
  Vd hàm tự định nghĩa:

```
def cong(x,y):
    return x+y
```

b) Cấu trúc tổng quát của hàm và cách gọi hàm:

```
def nameFunction(đối1,đối2,...):
    các lệnh của hàm
```

- Quy tắc đặt tên hàm:
  - snake_case → ten_ham()
  - camel_case → tenHam()
- Nguyên tắc hoạt động của hàm:"Vào sau ra trước"
- Các hàm có thể có đối số hoặc không hoặc là tham số mặc định
- Kiểm tra có đối số hay có kết quả trả về không
  - Nếu có: Result = FunctionName([parameter]) →Nếu có return
  - Nếu không: FunctionName([parameter]) → Nếu có print(ko return)

vd:

```
def ptb1(a,b):
    if a == 0 and b == 0:
        return "Vô số nghiệm"
    elif a == 0 and b != 0:
        return "Vô nghiệm"
    else:
        return "x = {0}".format(round(-b/a,2))
→ Trả về kết quả
kq = ptb1(5,8)
print(kq)
```

```
def xuatdulieu(data):
    print(data)
→Không trả về kết quả
Xuatdulieu("Hello")
```

c) Viết tài liệu cho hàm:

- Có thể dùng """ để viết tài liệu cho hàm (3 dấu nháy kép)
- Ghi chú phải viết những dòng đầu khi khai báo hàm
- Muốn xem tài liệu hàm gõ: 'help(tên hàm)'

vd:

```
def gcd(n1,n2):
    """Hàm này dùng để tìm UCLN"""
    min = n1 if n1 < n2 else n2
    largest_factor = 1
    for i in range(1,min +1):
        if n1 % i == 0 and n2 % i == 0:
            largest_factor = i
    return largest_factor
```

```
def ptb1(a,b):
    """Giải phương trình bậc nhất"""
    if a == 0 and b == 0:
        return "Vô số nghiệm"
    elif a == 0 and b != 0:
        return "Vô nghiệm"
    else:
        return "x = {0}".format(round(-b/a,2))
```

d) Global Variable:

- Các biến khai báo trong hàm chỉ có phạm vi trong hàm gọi là biến local → khi thoát ra khỏi hàm thì các biến này không thể truy suất được

vd:

```
g = 5           #Global variable
def incremant():
    g = 2           #Local variable
    g += 1
increment()
print(g)        #Global variable
→KQ: 5
```

```
g = 5
def increment():
    global g        #Thay đổi g = 5
    g = 2
    g += 1
increment()
print(g)
→KQ: 3
```

```
g = 5
def increment():
    g += 1
increment()
print(g)
→Lỗi vì lấy g trong hàm
không lấy ở ngoài
```

- Biến local chỉ chạy trong hàm def, chỉ khi chèn global vào def mới chạy ra ngoài chương trình chính
- Riêng đối với Collection vừa đóng vai trò Global và Local

e) Parameter mặc định:

```
def print(self, *args, sep=' ', end='\n', file=None): #
    """
    print(value,..., sep=' ', end='\n', file=sys.stdout)
```

Vd1:

```
def SumRange(n,m=0):            #Nếu không gọi m mặc định là 0
    sum = 0
    for i in range(1,m+n,1):
        sum = i
    return sum

x1=SumRange(5)
print('x1=',x1)                 → x1 = 4
x2=SumRange(5,1)
print('x2=',x2)                 → x2 = 5
```

Vd2:

```
def tinhTong(*nameVariable):
    tong = 0
    for item in num:
        tong += item
    return tong
```

→ đầu vào vô hạn, không biết trước số lượng

f) Lambda expression:

- Khai báo hàm nặc danh thông qua Lambda expression:

```
lambda parameterlist :    expression
```

- lambda: là từ khóa
- parameterlist: tập hợp parameter mà ta muốn định nghĩa (các biến)
- expression: biểu thức đơn trong Python

vd:

#Chi tiết:

```
def handle(f,x):
    return f(x)          #Đối số 1 là 1 hàm f nào đó
ret = handle(lambda x: f(x), x)

ret1=handle(lambda x: x%2==0, 7)
ret2=handle(lambda x: x%2 !=0, 7)
```

hoặc thay trực tiếp vào các hàm

```
def handle(f,x):
    return f(x)
def sochan(x):
    return x%2==0
def sole(x):
    return x%2 != 0
ret1=handle(sochan,6)
ret2=handle(sole,7)
```

→ Trước dấu 2 chấm là từ khóa lambda, đằng sau nó là số lượng các biến được khai báo trong handle (tính sau chữ f). Tức nếu có handle(f,x,y) thì viết lambda x,y:

```
def handle(f,x,y):
    return f(x,y)
sum=handle(lambda x,y: x+y, 7, 9)        #7 thay thế x, 9 thay thế y
print(sum)
```

g) Điểm khác nhau giữa các cách sử dụng hàm def trong Python:

def ten_ham(tên biến): kq = các phép tính return kq1, kq2, ... giatri1, giatri2, ... = ten_ham(tên biến) print(giatri)	def ten_ham(tên biến): kq = các phép tính print(kq) ten_ham(tên biến)

h) <u>Sơ lược hàm đệ quy:</u>

- Đệ quy là cách mà hàm tự gọi chính nó trong TH hóc búa —> Xử lý khéo
- Nếu dùng vòng lặp không được thì dùng đệ quy
- Ứng dụng đệ quy: tính giai thừa, tính dãy Fibonacci
  - n! = n*(n-1)! —> Đệ quy: biết được (n-1)! sẽ tính được n!
  - f1 = 1, f2 = 1, fn = f(n-1) + f(n-2)
- Phải có điểm dừng bài toán và quy luật thực hiện bài toán

vd: Đệ quy tính giai thừa:
n! = n(n-1)(n-2)...3.2.1

—>Dạng đệ qui: n! = 1, if n =0 n(n-1)! => Điểm dừng là khi n = 0, quy luật là nếu biết (n-1)! thì tính được n!	def factorial(n): if n == 0: return 1 else: return n *factorial(n-1) kq = factorial(6) print(kq)

Bài 14: Module quản lý

1. **Khai báo Module:**

```
import module1, module2, ... as <tên viết tắt>
```

- Phân hóa chương trình bằng ra các nhánh nhỏ để dễ quản lý, gọi lại khi cần và tái sử dụng, có tính bảo trì cao
- Trong đó: module1, module2, ... là các modules mà muốn import vào file hiện tại
- Ta có thể gọi tên tắt của module thư viện sau từ khóa "as"

2. **Cấu trúc Module:**

```
from modules import something1, something2, ...
```

- Modules là tên của module mà muốn import
- something1, something2, ... là những thứ muốn sử dụng trong modules (như các function1, function2, ...)
- Nếu muốn import tất cả mọi thứ trong modules thì sử dụng *

3. **Module Folder và File:**

```
from package.nameFile import nameFunction

if __name__ == "__main__":
    nameKQ = nameFunction(val1, val2, ...)
```

- from → chứa các module (tên folder, tên file)
- import → chứa các hàm function trong module
- Chỉ có tác dụng lấy các function trong ***package.nameFile***

4. **Sơ lược qua về thư viện xử lý dữ liệu trên terminal:**
   - Giúp tương tác với các function lớp dưới của hệ điều hành (tên người dùng, tên hệ điều hành, các biến môi trường)
   - Lấy các thông tin cơ bản trong hệ thống:
     - Liệt kê các thư mục và file
     - Tạo/xóa/đổi tên các thư mục hoặc file
     - Thực thi các lệnh command line hoặc terminal
   - Xuất thư viện OS ra: import os
     - Tên hệ điều hành: os.name
     - Câu lệnh xóa terminal: os.system("cls")
     - Thực thi lệnh hệ thống: os.system("cmd")
     - Đổi tên file: os.rename("<oldName>", "<newName>")

- Xóa file: os.remove("<tên file>")
- Xóa folder rỗng: os.rmdir()

- Lấy tên đăng nhập: os.getlogin()
- Lấy đường dẫn folder hiện tại: os.getcwd()
- Liệt kê file và folder trong cwd: os.listdir()
- Lấy biến môi trường environ: os.environ.get("PATH")

- Kiểm tra Module OS:
  - os.path.exists(path): Kiểm tra đường dẫn có tồn tại không
  - os.path.isdir(path): Kiểm tra đường dẫn có phải là 1 thư mục không
  - os.path.isfile(path): Kiểm tra đường dẫn có phải là 1 file hay không
  - os.mkdir: Tạo thư mục mới
  - os.makedirs("dir1/dir2"): Tạo thư mục với thư mục con

  - os.path.join(folder, file): Nối đường dẫn
    - newCurrentFolder = os.path.join(oldCurrentFolder, newFolder)

  - os.path.basename(folder/file): Lấy tên file từ đường dẫn
  - os.path.dirname(folder/file): Lấy tên đường dẫn

- Xuất thư viện platform ra: import platform
  - Tên của OS System: platform.system()
  - Phiên bản version: platform.release()

- Xuất thư viện shutil: import shutil
  - Cú pháp: shutil.copy2("source_file", "destination")
    - Tính chất:
      - Di chuyển file từ thư mục này sang thư mục khác
      - copy2 cho phép copy cả các thông tin về metadata, permissions
      - Đối số destination trong copy2 có thể là đường dẫn tới thư mục

    - Thư viện bảng:

Function	Copies metadata	Copies permissions	Can use buffer	Dest dir OK
shutil.copy	No	Yes	No	Yes
shutil.copyfile	No	No	No	No
shutil.copy2	Yes	Yes	No	Yes
shutil.copyfileobj	No	No	Yes	No

  - Cú pháp: shutil.move("source_file", "destination")
    - Tính chất:
      - Di chuyển file từ thư mục này sang thư mục khác
      - Tương tự os.rename()

Bài 15: Thư viện numpy, matplotlib, pandas

a) Thư viện numpy:

Thư viện NumPy là thư viện cốt lõi cho tính toán khoa học bằng Python. Nó cung cấp một đối tượng mảng đa chiều hiệu suất cao và các công cụ để làm việc với mảng.

```
import numpy as np
```

+ Gọi ma trận:

```
lst = np.array([a,b,c])          #Mảng 1 chiều
lst = np.array([(a,b,c,d), (x,y,z,t)]   #Mảng 2 chiều
```

+ Kích cỡ ma trận: lst.shape

+ Loại đối tượng: type(_) hoặc np.object()

+ Số chiều: lst.ndim

+ Số phần tử trong mảng: lst.size

+ Kiểu số các phần tử trong mảng: lst.dtype

+ Ép kiểu các phần tử thành số nguyên trong mảng: lst.astype(_)

- Các dạng ma trận đặc biệt:
  - Ma trận 0: np.zeros(a,b)
  - Ma trận 1: np.ones(a,b)
  - Mảng gồm các giá trị cách đều nhau với bước nhảy: np.arange(a,b,"bước nhảy")
  - Mảng cách đều khoảng cho trước: np.linspace(a,b,"khoảng")
  - Ma trận đơn vị: np.eye(n)
  - Ma trận chuyển vị: np.transpose(lst) hoặc lst.T
  - Mảng ngẫu nhiên: np.random.random((a,b))
  - Độ dài mảng: len(lst)
- Các kiểu dữ liệu numpy:
  - np.int64: 64bit số nguyên
  - np.float32: số thực
  - np.complex: số phức
  - np.bool: toán tử True/False
  - np.string_: loại chuỗi có độ dài cố định
  - np.unicode_: loại unicode có độ dài cố định
- Các phép tính trong numpy:
  - np.add(a,b): phép cộng (a+b)
  - np.subtract(a,b): phép trừ (a-b)
  - np.multiply(a,b): phép nhân (a*b)

        - np.divide(a,b): phép chia (a/b)
    - Các hàm số cơ bản trong numpy:
        - np.exp(x): hàm e^x
        - np.sqrt(x): hàm căn bậc hai
        - np.sin(x): hàm lượng giác sin(x)
        - np.cos(x): hàm lượng giác cos(x)
        - np.log(x): hàm logarit(x)
        - …
    - Chức năng tính toán thống kê:
        - lst.sum(): tính tổng các phần tử trong mảng
        - lst.min(): giá trị bé nhất trong mảng
        - lst.max(): giá trị lớn nhất trong mảng
        - lst.cumsum(): Tổng tích lũy các phần tử
        - lst.mean(): giá trị trung bình của mảng
        - lst.media(): giá trị trung vị, số đứng ở giữa
        - lst.corroef(): hệ số tương quan
        - np.std(): độ lệch chuẩn
    - Các phương thức trên ma trận numpy:
        - Thay đổi kích cỡ ma trận:
            - lst.ravel(): biến thành 1 hàng
            - lst.reshape(a,b): biến ma trận cỡ bxa thành cỡ axb
        - Chèn các phần tử: np.insert(lst, i, a) #chèn a vào vị trí i
        - Xóa bỏ phần tử: np.delete(lst, i) #xóa ptu I trong lst
        - Kết hợp mảng: np.concatenate((lst1,lst2), axis = 0))
        - Sắp xếp mảng:
            - Theo chiều dọc: np.vstack((a,b))
            - Theo chiều ngang: np.hstack((a,b))
        - Tách mảng:
            - Theo chiều ngang: np.hsplit(lst)
            - Theo chiều dọc: np.vsplit(lst)

b) <u>Vẽ đồ thị Matplotlib:</u>

+ Matplotlib là thư viện tạo hình 2D trong Python, thiết kế và tạo đồ thị đúng với yêu cầu
+ Có thể sửa đổi nhiều thứ trong plot gồm: giới hạn, màu sắc, ticks nhãn, tiêu đề
+ Cung cấp nhiều dạng đồ thị khác nhau
+ Kết hợp với bộ thư viện numpy, pandas

```
import matplotlib.pyplot as plt
import numpy as np
```

- Thực hiện các phép toán qua numpy bởi các hàm qua x và y
- In ra màn hình: plt.plot(x,y) → dạng đồ thị

```
import matplotlib.pyplot as plt
```

```
import numpy as np
X = np.linspace(0, 2*np.pi, 1000)      #Đồ thị y = X là đườn thẳng trong kh
oảng [0,2pi]
Y = np.sin(X)      #Đồ thị y = sin(x) trong khoảng [0,2pi]
plt.plot(X,Y, …)
plt.show()
```

- Các phương thức khác: (ghi luôn vào trong plot)
  - Đổi màu dòng kẻ: c = "màu"
  - Loại đường: linestyle = "-"
  - Băng thông: linewidth = 2
  - Ghi chú biểu đồ: label = "tên hàm"
  - …
- Các dạng đồ thị đặc biệt: plot(dạng kẻ), scatter(dạng chấm), bar(dạng cột), imshow(dạng ảo che), countor(dạng phân phối), pie(biểu đồ tròn), hist(biểu đồ cột), ….

c) <u>Thư viện xử lý Pandas:</u>

+ Cách import thư viện: import pandas as pd

+ Kết hợp thêm thư viện numpy và matplotlib để phân tích dữ liệu mpg

- Đặt tên trục:
  - Trục Ox: plt.xlabel("text", fontsize = phông chữ)
  - Trục Oy: plt.ylable("text", fontsize = phông chữ)
- Điều chỉnh khoảng các trên trục: plt.yticks([a, b, c, …])
- Xóa bỏ dữ liệu dư thừa ngoài biểu đồ: plt.grid()
- Thêm chú thích ô nhỏ: plt.legend()
- Lưu ảnh biểu đồ: plt.save("my_fig.jpf", dpi = khoảng chia)

+ Đọc dữ liệu và phân tích:

- Load data (đọc dữ liệu):

  data = pd.read_csv("filename.csv")

  data.head()

- Nhóm dữ liệu lại:

  Date.groupby(by = "tên cột", as_index = False).Phép_toán

- Cách gọi các trường thông tin:

  Mpg = data['tên cột']

- In ra dữ liệu từ cột trường đã lấy ra: mpg.describe()

d) <u>Thư viện sử dung hàm có sẵn Sympy:</u>

VD: fibonacci() → tự động tính dãy Fibonacci

Bài 16: Xử lý tập tin dữ liệu

a) Cách ghi tập tin:

- Mở file để ghi:
  - open('myfile.txt','w',encoding = 'utf-8') mở tập tin để ghi mới #Dữ liệu cũ xóa
  - open('myfile.txt',a',encoding = 'utf-8') mở tập tin để ghi nối đuôi # Dữ liệu cũ không bị xóa

- Ghi file:
  - file.write("contentString") → ghi liền nội dung vào file
  - file.writelines("contentString") → ghi file từng dòng

- Hàm lưu file:

```
def luufile(path):
    file = open(path,'w',encoding='utf-8')            #Ghi kiểu mới
    file.writelines("SV001;Tạ Quang Tùng;6/3/2003")
    file.writelines("SV002;Tạ Như Quỳnh;9/1/2001")
    file.close()
luufile("csdl.txt")
```

→ Câu lệnh chốt của hàm phải có file.close()

b) Cách đọc tập tin:

- Mở file để đọc:
  - open('myfile.txt','r',encoding = 'utf-8')

- Đọc file:
  - Content = file.read() → đọc tất cả nội dung trong file
  - Line = file.readline() → đọc từng dòng của file
  - Content1 = file.read(<size>) → lấy số lượng <size> ký tự đầu tiên

- Câu lệnh chốt đọc file xong → file.close()

- Có thể duyệt file đó bằng vòng lặp

C1: Đọc file qua hàm def

```
def docfile(path):
    file = open(path,'r',encoding='utf-8')        # Mở file
    for line in file:
        data = line.strip()
        print(data)
    file.close()          # Đóng file

docfile("csdl.txt")
```

C2: Đọc file qua cấu trúc with … as … :

```
path = 'data.txt'
```

```
with open(path, 'r') as f:
    allLines = f.read().splitlines()
   allLines = list(f)
```

C3: Đọc file không cấu trúc:

```
f = open('data.txt', 'r')
allLines = f.read().splitlines()
f.close()
```

c) Bảng từ khoá cho đọc và ghi file:

Module	Mô tả
r	Mở file chỉ để đọc
r+	Mở file để đọc và ghi
rb	Mở file trong chế độ đọc cho định dạng nhị phân (mặc định), con trỏ tại phần bắt đầu của file
rb+	Mở file để đọc và ghi trong định dạng nhị phân, con trỏ tại phần bắt đầu của file
w	Tạo một file mới để ghi, nếu file đã tồn tại thì sẽ bị ghi mới
w+	Tạo một file mới để đọc và ghi, nếu file tồn tại thì sẽ bị ghi mới
wb	Mở file trong chế độ ghi trong dạnh nhị phân, nếu file đã tồn tại thì ghi đè nội dung của file đó, nếu không thì tạo một file mới
wb+	Mở file để đọc và ghi trong định dạng nhị phân, nếu file tồn tại thì ghi đè nội dung của nó, nếu file không tồn tại thì tạo một file mới để đọc và ghi
a	Mở file để ghi thêm vào cuối file, nếu không tìm thấy file sẽ tạo mới một file để ghi mới
a+	Mở file để đọc và ghi thêm vào cuối file, nếu không tìm thấy file sẽ tạo mới một file để đọc và ghi mới
ab	Mở file trong chế độ append trong chế độ nhị phân (con trỏ ở cuối file nếu file tồn tại). Nếu file không tồn tại thì tạo một file mới để ghi
ab+	Mở file để đọc trong chế độ append trong chế độ nhị phân (con trỏ ở cuối file nếu file tồn tại). Nếu file không tồn tại thì tạo một file mới để đọc và ghi

d) Thuộc tính của File:

- Trả về True nếu file đã đóng, ngược lại là False → ***file.closed***
- Trả về chế độ truy cập của file đang được mở → ***file.mode***
- Trả về tên của file → ***file.name***

e) Một số Keyword Arguments:

*args	**kwargs	argv
+ có thể truyền nhiều tham số mà không cần định nghĩa trước đó `def my_func(*args):` `  return sum(args)` `print("Hàm args:",my_func(1,2,3))` `print(my_func(1,2,3))` `#không giới hạn số biến đầu vào`	+ có thể đổi tên biến tham số truyền vào thành 1 dictionary theo nguyên tắc key = value + kwargs.items() gồm cả key và value Hoặc kwargs.keys() → gọi key Hoặc kwargs.values() → gọi value + thiếu 1 thành phần thì bị lỗi `def funct(**kwargs):` `  print(kwargs)` `funct(c=1,e=2)` → `{'c': 1, 'e': 2}`	+ chạy trên terminal với file py `from sys import argv` + Cách truyền tham số từ vị trí thứ 1 trở đi a = argv[1:] + Viết đầu vào trực tiếp lên terminal

- **Chú ý: Chỉ thực hiện trên terminal**

f) Unpack * và **:

+ Cho a = [list ban đầu]
  - ➔ *a là in ra các "key" hoặc in ra từng phần tử trong list
  - ➔ **a là in ra các "value" của list

g) Hàm xóa dữ liệu truncate(0):

+ Xóa sạch nội dung trong file dạng txt

```
f = open("filename.txt", 'r+')
f.truncate(0)
```

Bài 17: Lập trình hướng đối tượng OOP

a) Kiến thức cơ bản về lớp Class và đối tượng Object:

```
class Object:
  # Khởi tạo Constructor các thuộc tính
  def __init__(self, name, …):
    self.name = name
    self.object1 = object1
    …
    print("Text")
  #Phương thức thực hiện (Method)
  def Tenham1(self):
    print("Tex1")
  def Tenham2(self, ..):
    print(f"Hello! My name is {self.name}")
  ..
"""gọi hàm khởi tạo đối tượng"""
obj1 = Object(name = "Tung")    #đối tượng A
..
"Lấy hàm trong phương thức của class" #SD thuộc tính in Method
#Cách 1:
obj1.Tenham1()
obj2.Tenham2()
..
#Cách 2: Object.Tenham(obj)
```

- Các thuộc tính trong một lớp thực chất là các biến → Đặt tên là danh từ
- Các phương thức trong một lớp thực chất là các hàm (luôn có tham số mặc định truyền vào là self) → Đặt tên là động từ

- Trông Python thì Số nguyên, Số thực, Chuỗi, List hay Dict đều là đối tượng

b) Thuộc tính và phương thức của lớp:

- Công thức tổng quát:

```
class Đối_tượng:
    # Khai báo thuộc tính mặc định
     t1 = <value1>              # Thuộc tính 1
     t2 = <value2>              # Thuộc tính 2

     # Khởi tạo Constructor
     def __init__(self, key1, key2, …):
         self.key1 = key1
```

```
        self.key2 = key2
        …

    # self = self.key1, self.key2, …      → thuộc tính

    # Khởi tạo các hàm getter và setter
    def set_key1(self, key1):
        self.key1 = key1
    def get_key1(self):
        return self.__key1

    def set_key1(self, key1):
        self.key1 = key1

    def get_key1(self):

        return self.__key2

    ...

    # Khai báo các phương thức
    def tên_hàm(self):
        # Xử lý các dữ liệu trong chương trình
        return kq           # Cách 1
        print(kq)           # Cách 2

# Khởi tạo đối tượng có tham số (có __init__)
DT1 = Đối_tượng(key1, key2, …)
# Khởi tạo đối tượng mặc định (không dùng __init__)
DT2 = Đối_tượng()

# Khai báo thuộc tính
+ Đối_tượng.t1, Đối_tượng.t2, … → Thuộc tính chung của toàn đối tượng
+ DT1.key1, DT1.key2, … → Thuộc tính của đối tượng DT1
+ DT1.t1, DT1.t2, … → Thuộc tính của DT1 từ thuộc tính chung

# In ra màn hình các phương thức của đối tượng
Cách 1:         print(DT1.tên_hàm())    → print
Cách 2:         DT1.tên_hàm()           → return

# Gán giá trị cho các thuộc tính (đối tượng mặc định - không tham số)
DT1.key1 = <value1>
DT1.key2 = <value2>
```

- Lưu ý:
    - Chỉ với các loại đối tượng mặc định (không có tham số) thì mới có thể gán giá trị ở bên ngoài class

- Trong Class luôn luôn có các Constructor, hàm getter và setter
- Có thể dụng phương thức property(get_name, set_name) thay cho các hàm getter và setter

Vd: Thuộc tính lớp Sieunhan

#Khai báo và sử dụng biến trong Class

```
class Sieunhan:
    # Khai báo giá trị mặc định cho thuộc tính
    stt = 0
    sothutu = 0          #của toàn lớp
    suc_manh = 50
    def __init__(self, para_ten, para_vukhi, para_mausac):
        self.ten = "Siêu nhân " + para_ten
        self.vukhi = para_vukhi
        self.mausac = para_mausac
        self.stt = Sieunhan.sothutu
        Sieunhan.sothutu += 1
SieunhanTT = Sieunhan("Kteam", "Knowlegde", "Green Blue")

#Cách gọi biến từ class
print(Sieunhan.suc_manh)            #gọi biến suc_manh từ Class
print(SieunhanTT.suc_manh)          #gọi biến suc_manh của TT bị tđ
```

#Cập nhập giá trị thuộc tính thông qua lớp

```
Sieunhan.suc_manh = 40
print(Sieunhan.suc_manh)
print(SieunhanTT.suc_manh) #Toàn bộ đối tượng trong Class đều bị cập
nhập lại
→40 40
#Thứ tự siêu nhân sau khi cập nhập giá trị
SieunhanA = Sieunhan("Siêu nhân Đỏ", "Kiếm", "Đỏ")
SieunhanB = Sieunhan("Siêu nhân Xanh", "Dao găm", "Xanh")
print(SieunhanA.stt)    →1
print(SieunhanB.stt)    →2
print(Sieunhan.sothutu) →3
```

#Cập nhập giá trị thuộc tính thông qua đối tượng

```
print(Sieunhan.suc_manh)       →40
print(SieunhanTT.suc_manh)     →40
SieunhanTT.suc_manh = 30      #Chỉ thay đổi 1 đối tượng nhỏ trong Class
print(Sieunhan.suc_manh)       →40
print(SieunhanTT.suc_manh)     →30  #Chỉ có TT bị thay đổi
```

c) Các phương thức lớp trong Method:

- Các phương thức đặc trưng trong OOP Python

@classmethod	@staticmethod
+ Sử dụng độc lập với dữ liệu trong Class + Thuộc tính ở lớp cũng như tất cả đối tượng thuộc lớp đó sẽ tự cập nhập giá trị mới + Công thức chung:	+ Xử lý vấn đề khác và có chức năng như 1 hàm bình thường `class Sieunhan:` `  sucmanh = 50`

```
class Đối_Tượng:
    def __init__(self, key1, key2, ..):
        self.key1 = key1
        self.key2 = key2
        ...
    @classmethod
    def  tên_hàm(self, stringle):
        Các tính toán trong hàm

# Khai báo trong classmethod
DT = Đối_Tượng(key1, key2, ...)
Dtmethod = Đối_Tượng.tên_hàm(stringle)
```

Vd: Phương thức classmethod

```
class Sieunhan:
    sucmanh = 50
    def __init__(self, para_ten, para_vukhi,
para_mausac):
        self.ten = para_ten
        self.vukhi = para_vukhi
        self.mausac = para_mausac
    @classmethod
    def from_string(cls, s):  #thường những phương
thức xử lý thế này hay có tên là from...
        lst = s.split('-')
        new_lst = [st.strip() for st in lst]
        ten, vukhi, mausac = new_lst
        return cls(ten, vukhi, mausac)
infor_str = "xanh - Kiếm - Xanh"
sieu_nhanA = Sieunhan.from_string(infor_str)
print(sieu_nhanA)
```

```
    def __init__(self, para_ten,
para_vukhi, para_mausac):
        self.ten = para_ten
        self.vukhi = para_vukhi
        self.mausac = para_mausac
    @staticmethod
    def bienhinh():
        print("1,2,3 Siêu nhân biến hình")
sieu_nhan_A = Sieunhan("Siêu nhân đỏ",
"Kiếm", "Nước")
print(sieu_nhan_A)
```

```
@staticmethod
def bienhinh():
    ...
```

Hoặc

```
def bienhinh(self):
    ...
```

- Nếu dựng 1 phương thức cần sử dụng đối tượng đó → regular method
- Nếu dùng dữ liệu có class → class method
- Có dùng đến dữ liệu nhưng không truyền gì cả → static method

d) Tạo lớp kế thừa Inheritance:

+ Inheritance có tính chất đề cập đến việc định nghĩa một lớp mới newClass dựa trên một lớp cũ đã cho existingClass (lớp con và lớp cha):
    - Lớp mới là lớp dẫn xuất (derived class) → lớp con (child class)
    - Lớp cũ là lớp cơ sở (base class) → lớp cha (parent class)

+ Công thức của lớp Kế thừa – Python nâng cao

```
class Đối_Tượng:
    def __init__(self, key1, key2, ...):
        self.key1 = key1
        self.key2 = key2
```

```
        …
    def ham(self):
        Các phương thức tính toán

class Kế_Thừa(Đối_Tượng):
      # Định nghĩa lại hàm khởi tạo trong class kế thừa (overriding)
      def __init__(self, key1, key2, …, val1, val2, …):
          super().__init__(key1, key2, …)      # Cách 1
          Đối_tượng.__init__(key1, key2)       # Cách 2
          self.val1 = val1
          self.val2 = val2
          …

    def hamKetthua(self):
        return super().ham()
TT = Kế_Thừa(key1, key2, …, val1, val2, …)
```

Vd: Lớp kế thừa trong OOP

#Lớp kế thừa Inheritance là tạo lớp mới kế thừa bổ sung cho lớp cũ

```
class Sieunhan:          #Lớp cũ
    sucmanh = 50
    suc_manh = 50
class SieunhanKteam(Sieunhan):      #Lớp mới bổ sung kế thừa từ lớp cũ
    suc_manh = 40
Kteam_do = SieunhanKteam()
print(Kteam_do)
print(Kteam_do.sucmanh)          #Lớp mới kế thừa sucmanh từ lớp cũ
print(Kteam_do.suc_manh)         #Lớp mới thay đổi không kế thừa lớp cũ
```

#Kế thừa hàm constructor bằng cách sử dụng: super().__init(…)

```
class Sieu_nhan:          #Lớp cũ
    sucmanh = 50
    def __init__(self, para_ten, para_vukhi, para_mausac):
        self.ten = para_ten
        self.vukhi = para_vukhi
        self.mausac = para_mausac
    def show(self):           #Hàm show từ lớp cũ
        print("Hello Tùng")
class Sieunhanteam(Sieu_nhan):
    sucmanh = 40
    def __init__(self, para_ten, para_vukhi, para_mausac, para_suthu):
#Vừa kế thừa vừa được thêm
        """
        self.ten = para_ten
        self.vukhi = para_vukhi
        self.mausac = para_mausac
        self.suthu = para_suthu
        """
        # Tái sử dụng tối ưu hơn: super().__init__(...)
        super().__init__(para_ten, para_vukhi, para_mausac)
#super() là Sieu_nhan
```

```
        self.suthu = para_suthu
    def show(self):           #Hàm show của lớp mới
        print("Sức mạnh của ta là:", self.sucmanh)

gaoxanh = Sieunhanteam("Xanh", "Kiếm", "Green", "Cá mập")      #Thừa
hưởng của lớp cũ
print(gaoxanh.__dict__)              #Dạng list hướng đối tượng
print(Sieunhanteam.sucmanh)
```

#Kế thừa phương thức như kế thừa thuộc tính
+ Nếu lớp trước có phương thức gì → kế thừa toàn bộ
+ Nếu muốn thêm gì thì viết vào lớp kế thừa
+ Nếu muốn chỉnh sửa thì ta viết lại phương thức đó
Vd: gaoxanh.show() #Kế thừa phương thức trong hàm mới kế thừa

- Hàm __dict__ là dạng list của hướng đối tượng
- Một số hàm kiểm tra đối tượng khác:
  - isinstance(object, objectClass) → Kiểm tra một đối tượng có phải thực thể của một lớp hay không
  - issubclass(object1, object2) → Kiểm tra một lớp có phải lớp con của một lớp khác hay không
  - type(objectName) → Trả về kiểu dữ liệu của đối tượng

e) Tính đa hình - Polymorphism:

- Tính đa hình là một phương thức có thể thực hiện nhiều chức năng:
  - Có một phương thức khởi tạo:
    - Nếu là con chó sẽ kêu gâu gâu
    - Nếu là con mèo sẽ kêu meo meo
  - Các đối tượng này dùng chung một phương thức nhưng mà thực hiện các nhiệm vụ khác nhau tùy vào đối tượng đó

f) Các phương thức đặc biệt Special Methods:

+ Special Methods được quy ước sẵn tên
+ Định dạng chung là: __tên phương thức__

- __init__(self, …) → Module hướng đối tượng dạng ẩn
- __str__(self) → miêu tả rõ giá trị từ __init__ (ưu tiên khi dùng "print")
- __repr__(self) → miêu tả rõ thông tin chi tiết từ __init__ (chi tiết hơn str)
- __len__() → đếm chuỗi
- __add__() → tạo thêm đối tượng
- __getitem__(self, key) → lấy phần tử theo index → obj[key]
- __setitem__(self, key, value) → gán giá trị cho index → obj[key] = value
- __delitem__(self, key) → xóa phần tử theo index → obj[key]
- __contains__(self, item) → kiểm tra item có trong obj không
  Vd:

```
class SieuNhan:
    suc_manh = 50
    def __init__(self, para_ten, para_vu_khi, para_mau_sac):
        self.ten = para_ten
        self.vu_khi = para_vu_khi
        self.mau_sac = para_mau_sac
SN_A = SieuNhan('Sieu nhan Do', 'Kiem', 'Do')
print(SN_A)

def __str__(self):
    return 'Day la {}, su dung {}'.format(self.ten, self.vu_khi)
def __repr__(self):
    return 'ten: {}\nvu khi: {}\nmau sac: {}'.format(self.ten,
self.vu_khi, self.mau_sac)
print('%s' %SN_A)      #Lấy theo str
print('%r' %SN_A)      #Lấy theo repr

#Đếm chuỗi qua hàm len
s = "Tạ Quang Tùng"
print(len(s))        #Cách 1
print(s.__len__())   #Cách 2
def __len__(self):
    return len(self.ten)

#Tạo thêm đối tượng
print(int.__add__(2,3))      #Cộng số nguyên
print(str.__add__("Pisces ", "Kibo"))    #Cộng chuỗi
print(list.__add__([1,2], [3,4]))        #Cộng mảng
#Thêm đối tượng bằng gọi hàm
def __add__(self, mot_sieu_nhan_khac):
    return self.suc_manh + mot_sieu_nhan_khac.suc_manh
print(SN_A + SN_B)
print(SieuNhan.__add__(SN_A, SN_B))
```

g) Getter, Setter, Deleter:

Getter	Setter	Deleter
@property → biến methods thành thuộc tính (không cần dấu ()) + Đặt "@property" lên trước hàm def + Thay đổi giá trị của thuộc tính đối với những đối tượng chứa tham số	Setter là ngược lại của Getter, gán hàm mới vào __init__ + Đặt "@tên hàm.setter" trước hàm cần thêm vào __init__	Deleter → dùng xong rồi xóa luôn + Đặt "@tên hàm.deleter" phía trước hàm cần xóa + Gọi hàm ngoài class bằng cách del đối_tượngA.hàm_cần_xóa

- Ứng với mỗi biến private ta xây dựng hai phương thức get/set để nhập/xuất dữ liệu → biến __nameVariable với phương thức get_nameVariable và set_nameVariable ➔ Cặp getter và setter

h) Một số phương thức nâng cao của nhiều kiểu đối tượng:

+ Thực hiện cùng lúc với hai đối tượng:

```
class Ten_doi_tuong:
        def __init__(self, ten_bien):
            self.ten_bien = ten_bien
            …
        def ten_ham1(self, tham số,…):
            # Các phép tính cần sử lý
Dt1 = Ten_doi_tuong(ten_bienA)
Dt2 = Ten_doi_tuong(ten_bienB)
…..
Dt1.ten_ham1(Dt2,…, các tham số khác)
```

➔ Cũng có thể thực hiện phương thức với nhiều đối tượng khác nhau

i) Nạp chồng toán tử giữa các đối tượng: (Magic Methods)

```python
class Polynomial:
    def __init__(self, n):
        self.n = n
    def __add__(self, other):
        return (self.n + other.n)
    def __sub__(self, other):
        return (self.n - other.n)
    def __mul__(self, other):
        return (self.n * other.n)
    def __truediv__(self, other):
        return (self.n / other.n)
    def __floordiv__(self, other):
        return (self.n // other.n)
    def __mod__(self, other):
        return (self.n % other.n)
    def __pow__(self, other):
        return (self.n**other.n)
    def __neg__(self):
        return -self
    ...

polynomialA = Polynomial(n1)
polynomialB = Polynomial(n2)
```

```
#Phép cộng giữa hai đối tượng
polynomialC = polynomialA + polynomialB
print('polynomialC = polynomialA + polynomialB =', polynomialC)

#Phép trừ giữa hai đối tượng
polynomialD = polynomialA - polynomialB
print('polynomialD = polynomialA - polynomialB =', polynomialD)

#Phép nhân giữa hai đối tượng
polynomialF = polynomialA * polynomialB
print('polynomialF = polynomialA * polynomialB =', polynomialF.n)

#Số đối của đối tượng
polynomialE = -polynomialA
print("polynomialE = -polynomialA =", polynomialE)
```

- Còn có các phép toán khác giữa hai đối tượng:

– Các toán tử hai ngôi (Binary Operators):

Toán tử	Phương thức ma thuật (magic method)
+	__add__(self, other)
–	__sub__(self, other)
*	__mul__(self, other)
/	__truediv__(self, other)
//	__floordiv__(self, other)
%	__mod__(self, other)
**	__pow__(self, other)

– Các toán tử so sánh:

Toán tử	Phương thức ma thuật (magic method)
<	__lt__(self, other)
>	__gt__(self, other)
<=	__le__(self, other)

>=	__ge__(self, other)
==	__eq__(self, other)
!=	__ne__(self, other)

– Các toán tử gán:

Toán tử	Phương thức ma thuật (magic method)
-=	__isub__(self, other)
+=	__iadd__(self, other)
*=	__imul__(self, other)
/=	__idiv__(self, other)
//=	__ifloordiv__(self, other)
%=	__imod__(self, other)
**=	__ipow__(self, other)

– Các toán tử ba ngôi (Unary Operators):

Toán tử	Phương thức ma thuật (magic method)
–	__neg__(self, other)
+	__pos__(self, other)
~	__invert__(self, other)

j) Phạm vi truy cập thuộc tính và phương thức:

- 2 gạch là private --> ***__nameFunction() và __nameVariable***
- 1 gạch là protected --> ***_nameFunction() và _nameVariable***
- 0 gạch là public --> ***nameFunction() và nameVariable***

k) Các từ khóa Ngoại lệ/Cảnh báo:

- Từ khóa "raise":

- Hiển thị ra màn hình lỗi nếu điều kiện ngoại lệ đó xảy ra
- Cấu trúc: ***raise Exception("Ngoại lệ/Cảnh báo")***

- Từ khóa "assert":
  - Sử dụng khi nhất định điều kiện đó đáp ứng
  - Cấu trúc: assert <điều kiện>
  - Nếu không thỏa mãn điều kiện trong asert sẽ lỗi

Bài 18: Thiết kế giao diện Tkinter

a) Cách tạo cửa sổ với Tkinter:

- from tkinter import *
- Sau đó khai báo 1 biến: root = Tk() → đối với 1 cửa sổ
- Hoặc khai báo biến: root = Toplevel() → đối với nhiều cửa sổ
- Đặt tiêu đề: root.title("chuỗi")
- Cho phép thay đổi kích thước cửa sổ:
  root.resizable(height = True, width = True)
- Thay đổi kích thước tối thiểu:
  root.minsize(height=a, width=b)
- Hàm để hiển thị cửa sổ lên: root.mainloop()

b) Các control cơ bản trong Tkinter:

- Label: cho phép hiển thị thông tin nhưng không cho phép thay đổi
- Button: ra lệnh (muốn thoát không)
- Entry: nhập dữ liệu

=> Luôn gọi biết root = Tk() sau khi import()

```
from tkinter import *

#Khởi tạo đầu vào GUI object
root = Tk()
root.title("Học control cơ bản")           #Đặt tên
root.resizable(height=True, width=True)    #Cho phép thay đổi k/c
root.minsize(height=200,width=300)         #Chu vi min
root.maxsize(height=600,width=800)         #Chu vi max
root.geometry("axb")                  #Kích cỡ mặc định

#Tạo các Label
#Đưa Label vào tkinter
Label(root,text="Hello Tkinter",fg="Green").pack()
#fg là chọn màu sắc

#Tạo các Button
#Đưa Button vào tkinter
Button(root,text="Click me",command=root.quit).pack()
#Câu lệnh command=root.quit() là để có muốn thoát màn hình hay
không

#Tạo các Entries
#Đưa Entry vào tkinter
e = StringVar()                            #Cho phép nhập chuỗi vào
e.set("Tạ Quang Tùng")                    #Viết luôn chuỗi mặc định
vào
Entry(root,textvariable=e,width=30).pack()
root.mainloop()
```

- Label(root,text="chuỗi",fg="màu",font=("tohama",cỡ),justify=CENTER).grid(_)
  - grid(_) : hiển thị dạng lưới hợp nhất
  - trong grid(_) gồm row=a,column=b #columnspan/rowspan là trộn các cột/hàng vào nhau
  - Loại grid() có thể thay thế bằng loại place(relx = ?, rely = ?)
  - fg= " " : hiển thị màu
  - font = ("tohama",cỡ chữ)
  - StringVar () được sử dụng để chỉnh sửa văn bản của tiện ích con
- Entry(root,width=a,textvariable=hàm nào đó).grid(_)
- Button(_):
  - Để gọi button cần có hàm: frameButton=Frame()
  - Button(frameButton,text= "_", command=hàm nào đó).pack()
  - Nếu muốn thoát ra chèn thêm trong Button là command=root.quit
  - Nếu muốn xóa cửa sổ Window hiện tại thì dùng command = root.destroy()

c) Tạo boxlist trong Tkinter:

- Kết hợp giữa luufile() và docfile()
- Tạo list arr rồi thêm vào
- Chèn insert list đó vào listbox

VD:

```
#Chức năng hiển thị sách ra màn hình giao diện
def showsach():
    arrSach=docfile()
    listbox.delete(0,END)
    for item in arrSach:
        listbox.insert(END,item)
```

d) Các phương thức trong hàm Tkinter():

- name.pack() để thêm nó vào chính cửa sổ ấy
  - VD: frame.pack()
- name.pack_forget() để xóa tiện ich ấy ngay trên cửa sổ ấy
  - VD:

```
ten_ham_can_chuyen(root)
frame.pack_forget()
```

➔ Sử dụng frame.pack() trước khi sử dụng frame.pack_forget()

- Chỉ con trỏ chuột vào nút tương ứng bằng **listbox.curselection()**

➔ Sử dụng bằng vòng lặp for index

e) Cách sử dụng khác Tkinter qua cách gọi nhiều biến:

- import tkinter as tk
- Sử dụng các hàm con nếu có: ten_ham(root) và return lại frame đã khởi tạo khi kết thúc hàm
- Gọi biến cho Label, Entry, Button cho các hàm bằng cách:
  - frame = tk.Frame(root)
  - Gọi qua Label:

        - label1 = tk.Label(frame, text = “tên văn bản”, fg = “màu sắc”
        - label1.grid(row = hàng, column = cột)
    - Gọi qua Entries:
        - entry1 = tk.Entry(frame)
        - entry1.grid(row = hàng, column = cột)
    - Gọi qua Button:
        - btn1 = tk.Button(frame, text = “tên văn bản”, fg = “màu sắc”)
        - bt1.grid(row = hàng, column = cột)
- Phần chạy chính:
    - Có thể gọi thư viện vừa tạo với file mới:
      from filename import ten_ham
- root = tk.Tk()
- Tạo tiêu đề và kích cỡ giao diện bằng title() và geometry()
- Gọi các hàm ra từ tham số được truyền vào:
    - Frameham1 = ten_ham_1(root)
    - Frameham2 = ten_ham_2(root)
    - Ban đầu hiện ra đầu tiên: Frameham1.pack()
    - Chuyển hướng sang giao diện hai: Frameham2.pack_forget()

f) Cách tạo Menubar:

- Có thể tạo hàm chứa menubar() nếu muốn: def menubar(root):
- Gọi biến: menubar = tk.Menu(root)
- Tạo các giao diện của Menubar: file, edit, help, windows, …
    - file_menu = tk.Menu(menubar)
    - edit_menu = tk.Menu(menubar)
    - view_menu = tk.Menu(menubar)
    - help_menu = tk.Menu(menubar)
- Tạo các lệnh cho menu qua phương thức thêm:
    - file_menu.add_command(label = “Open”, command = lambda:
      ten_ham(các tham số))
- Hiển thị các menu:
    - Menubar.add_cascade(label = “File”, menu = file.menu)
- return menubar

g) Hộp thông báo thư viện Message:

- Xuất thư viện: from tkinter import messagebox
    - askokcancel(title, message, option)
    - askquestion(title, message, option)
    - askretrycancel(title, message, option)
    - askyesno(title, message, option)
    - showinfo(title, message, option)
    - showwarning(title, message, option)
    - showerror(title, message, option)

Bài 19: Giới thiệu về thư viện Pygame

a) Một số gợi ý trong thư viện Pygame:

+ Xuất các thư viện cần thiết như pygame, sys, …

+ Khởi tạo đối tượng game: pygame.init()

+ Tùy chỉnh giao diện kích cỡ màn hình hiển thị:

pygame.display.set_mode((width, height))

+ Chọn lựa màu sắc ngẫu nhiên qua bảng mã màu:

colors = [“các mã màu”, …]

color = random.choice(colors)

+ Thêm thời gian trong pygame: pygame.time.Clock()

+ Vòng lặp vĩnh cửu chính

b) Ví du minh hoa về Pygame:

```
#Xuất các thư viện cần thiết
import pygame
import sys
import random

#Khởi tạo đối tượng game và kích cỡ màn hình hiển thị
pygame.init()            #Khởi tạo đối tượng game

#Kích cỡ màn hình hiển thị
screenWidth = 500
screenHeight = 500
screen = pygame.display.set_mode((screenWidth, screenHeight))

#Tạo màu sắc ngẫu nhiên
colors = ['#C7980A',
          '#F4651F',
          '#82D8A7',
          '#CC3A05',
          '#575E76',
          '#156943',
          '#0BD055',
          '#ACD338']
color = random.choice(colors)

#Tọa độ x của đường tròn
coordinateX = 250
#Vận tốc của vòng tròn
```

```
velocity = 5

clock = pygame.time.Clock()          #Thời gian pygame

#Vòng lặp chính của game
while True:
    clock.tick(60)
    #Tạo event
    for event in pygame.event.get():
        #Xử lý trình thoát exit()
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    #Tô màu cho màn hình (màu xám)
    screen.fill((150, 150, 150))

    #Update tọa độ x của vòng tròn
    if coordinateX <= 75 or coordinateX >= 425:
        color = random.choice(colors)
        velocity *= -1
    coordinateX += velocity

    #Vẽ hình tròn màu dương giữa màn hình
    pygame.draw.circle(screen, color, (coordinateX, 250), 75)
    #Flip
    pygame.display.flip()
```

c) <u>Cách tải các gói dữ liệu thư viện package:</u>

- Tải package: ***pip install <package_name>***
  Vd: pip install pygame

- Hủy cài đặt package: ***pip uninstall <package_name>***
  Vd: pip uninstall pygame