# Chuyên cần: Học và làm bài tập

**14/11/2023: KTTX 1-Thực hành**

Lập trình và làm dc các dữ liệu: list, array, if for bla bla…

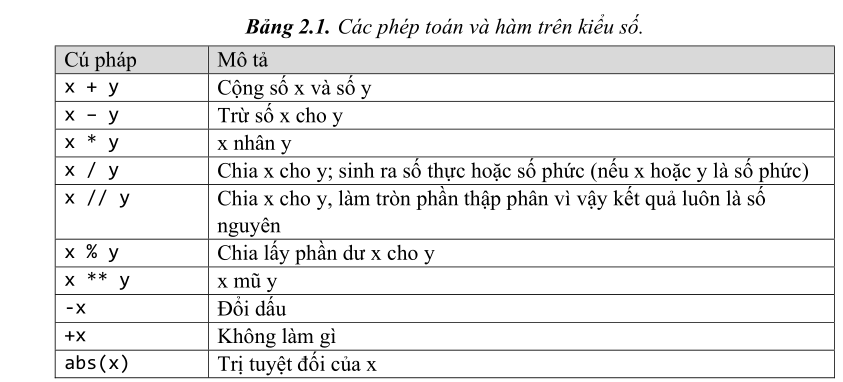
**12/12/2023: KTTX 2- Thực hành**

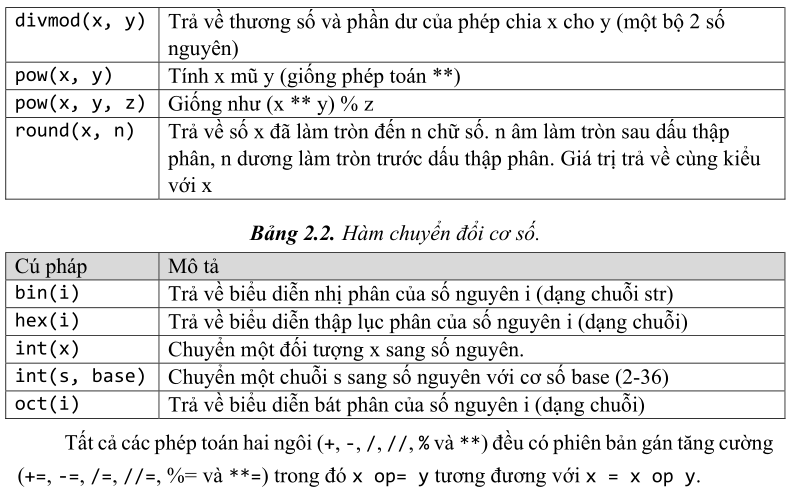
# Python: Beggin

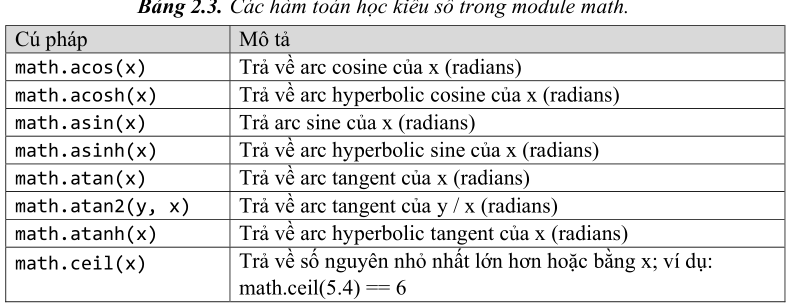
## Day 1: Khai báo

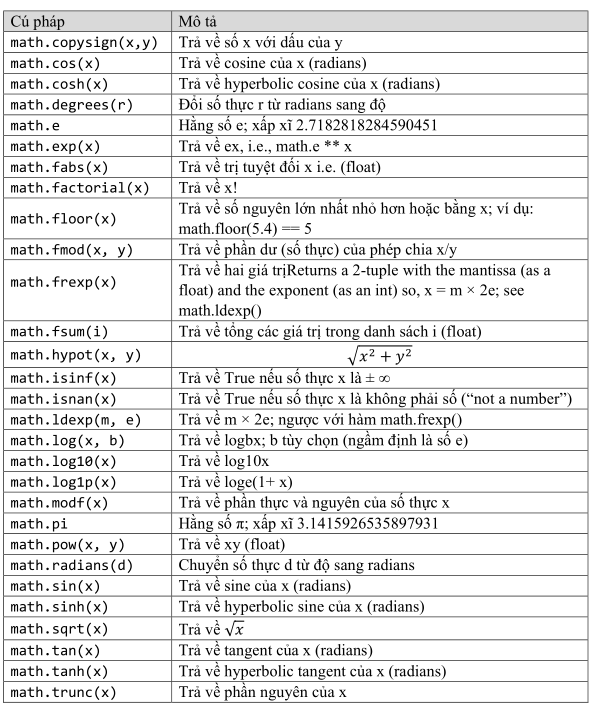
Cú pháp: print(“…”) => cmd: …

Print(“a+b =”,a+b) => a + b = a+b (trong python khi kb biến sẽ tự chèn khoảng trắng).









## Day 2: KIỂU DỮ LIỆU VÀ THAO TÁC NHẬP/ XUẤT

Int = 5000

Print(int)

Float

String = ‘ …‘

print (String)

Bool

Note:

Name = “hello”

Print(“xin chao ” + name)=>cmd: xin chao hello

Or Print(“xin chao ” \* 3) => xin chao xin chao xin chao

Bạn hãy viết chương trình nhập vào tên của mình từ bàn phím và thực hiện hiển thị ra màn hình dòng chữ.

…..

age = input()

print(type(age))

Nếu bạn nhập 19 và bấm phím Enter thì màn hình sẽ hiển thị ra:

<class 'str'>

--

Viết chương trình nhập từ bàn phím xâu x và kiểm tra xem trong xâu x có chứa ký tự 'H' hay không, nếu có thì hiển thị ra màn hình True, ngược lại hiển thị False.

x = input()

print('H' in x)

--

Lever 3:

--

Cho 2 số nguyên a và b được nhập từ bàn phím, bạn hãy viết chương trình hiển thị ra tổng các số lẻ từ a tới b.

Ví dụ nếu bạn nhập a = 3, b = 9 thì màn hình sẽ hiển thị ra:

3+5+7+9= 24

a = int(input())

b = int(input())

c= 0

# a=3, b=9 => i=3 c=3;i=4 c=3;i=5 c=3+5

for i in range(a,b+1):

if i%2!=0:

c+=i

print(c)

Lý thuyết

Bạn có thể sử dụng hàm round() để làm tròn các số thực. Cú pháp hàm round:

round(number, ndigits)

Trong đó number là số cần làm tròn, ndigits là số chữ số sau dấu phẩy cần làm tròn. Ví dụ:

print(round(1.23, 1))

print(round(2.665, 2))

print(round(2.673567, 4))

Kết quả khi chạy chương trình:

1.2

2.67

2.6736

Cho một list các số nguyên n phần tử lst được nhập từ bàn phím. Bạn hãy viết chương trình hiển thị ra màn hình số nhỏ nhất trong list vừa nhập.

Ví dụ

Nếu bạn nhập n = 3, lst = [1, 3, 2] thì chương trình sẽ hiển thị ra 1 (do 1 là giá trị nhỏ nhất trong lst):

Nếu bạn nhập n = 3, lst = [34, 35, 27] thì chương trình sẽ hiển thị ra 27.

Lý thuyết

list trong Python là một kiểu dữ liệu cho phép lưu trữ nhiều kiểu dữ liệu khác. Để khởi tạo một list trong Python bạn có thể sử dụng cặp dấu []. Ví dụ:

# Tạo ra list để lưu trữ các số nguyên

list1 = [1, 2, 3]

# Tạo ra list để lưu trữ các xâu ký tự

list2 = ["Viet", "Tuan", "Duong"]

# Bạn cũng có thể tạo ra một list lưu trữ các kiểu dữ liệu khác nhau

list3 = [7, 3.5, "Codelearn"]

Để truy xuất tới các phần tử trong list bạn dùng toán tử []. Ví dụ:

names = ["Viet", "Dung", "Huong"]

print(names[0])

print(names[1])

print(names[2])

Kết quả khi chạy chương trình:

Viet

Dung

Huong

Lưu ý: names[1] không phải là phần tử đầu tiên của list mà phải là names[0] (do chỉ số của list được bắt đầu từ 0).

Bạn cũng có thể dùng vòng lặp for để duyệt qua các phần tử của list. Ví dụ:

names = ["Viet", "Dung", "Huong"]

for name in names:

print(name)

Để thêm một phần tử vào cuối của list bạn dùng hàm append():

lst = []

lst.append(4)

lst.append(3)

lst.append(6)

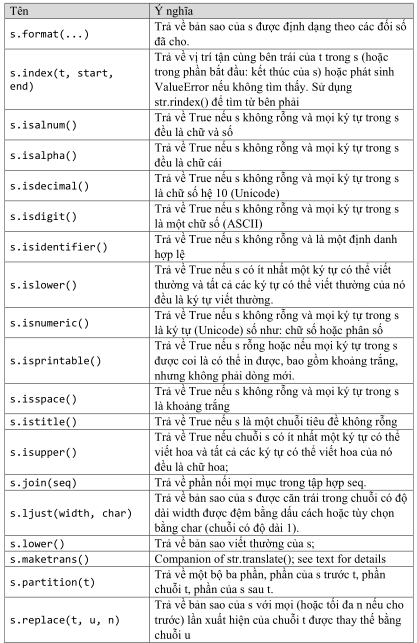
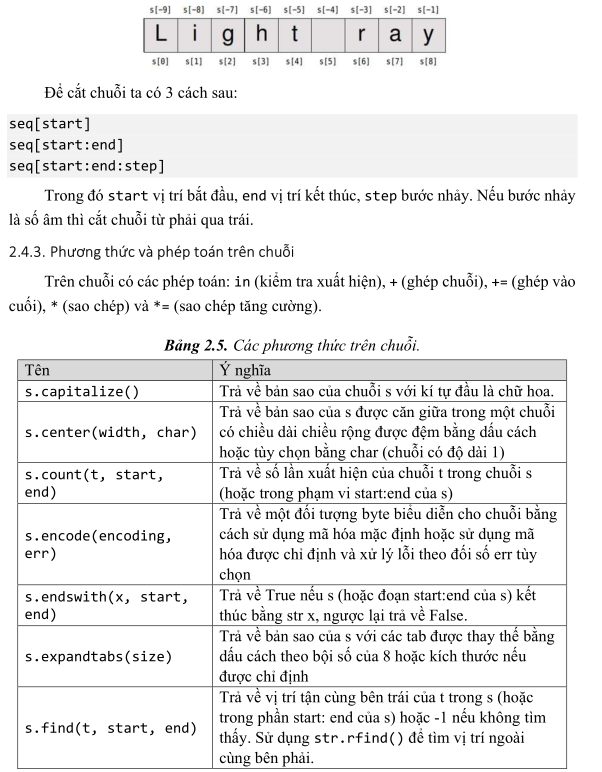
print(lst)

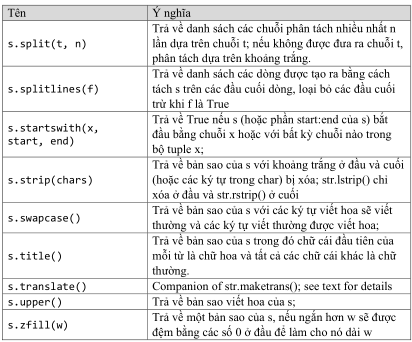
Kết quả khi chạy chương trình:

[4, 3, 6]

Đọc tới đây bạn đã biết cách sử dụng list trong python, hãy quay lại phần bài tập và làm thử.

### Các chức năng của biến thư viện có sẵn





## Giải thích

### 1. Hàm len

Đây là hàm trả về số phần tử có trong list. Ví dụ:

lst = [2, 3, 1]

print(len(lst))

Kết quả khi chạy chương trình:

3

Bạn có thể tận dụng hàm len() để hỗ trợ duyệt qua các phần tử trong list:

lst = [2, 3, 1]

for i in range(len(lst)):

print(lst[i])

### 2. Hàm max, min

Đây là hai hàm được dùng để trả về phần tử lớn nhất và nhỏ nhất trong list. Ví dụ:

lst = [2, 3, 1]

print(max(lst))

print(min(lst))

Kết quả khi chạy chương trình:

3

1

### 3. Hàm insert

Đây là hàm dùng để thêm một phần tử vào một ví trí trong list:

vowels = ['a', 'e', 'i', 'u']

# Chèn xâu 'o' vào vị trí thứ 4 trong list vowels

vowels.insert(3, 'o')

print(vowels)

Kết quả khi chạy chương trình:

['a', 'e', 'i', 'o', 'u']

### 4. Hàm remove

Hàm này dùng để xóa một phần tử khỏi list:

lst = ['A', 'B', 'C']

lst.remove('A')

print(lst)

Kết quả khi chạy chương trình:

['B', 'C']

### 5. Hàm pop

Hàm pop() được dùng để xóa một phần tử với chỉ số cho trước trong list:

lst = ['A', 'B', 'C']

# Xóa phần tử thứ 2 khỏi list

lst.pop(1)

print(lst)

Kết quả khi chạy chương trình:

['A', 'C']

### 6. Hàm sort

Hàm này được dùng để sắp xếp các phần tử trong list theo một thứ tự nhất định. Ví dụ:

lst = [4, 5, 3, 7, 6, 1]

# Sắp xếp các phần tử trong list theo thứ tự tăng dần

lst.sort()

print(lst)

# Sắp xếp các phần tử trong list theo thứ tự giảm dần

lst.sort(reverse=True)

print(lst)

Kết quả khi chạy chương trình:

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

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

### 7. Hàm reverse

Đây là hàm dùng để đảo ngược list:

lst = [4, 5, 3, 7, 6, 1]

lst.reverse()

print(lst)

Kết quả khi chạy chương trình:

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

### 8. Hàm count

Đây là hàm dùng để đếm số lần xuất hiện của một thành phần trong list:

lst = [6, 2, 3, 8, 2]

print(lst.count(2))

Kết quả khi chạy chương trình:

2

### 9. Hàm clear

Đây là hàm dùng để xóa hết các phần tử bên trong list:

lst = [1, 2, 3]

lst.clear()

print(lst)

Một số hàm và phương thức để xử lý chuỗi trong Python

### 10. Hàm len()

Đây là hàm trả về độ dài của chuỗi:

s = "abc"

print(len(s))

s = "abcd"

print(len(s))

Kết quả khi chạy chương trình: 3 4

### 11. Phương thức lower()

Đây là phương thức được dùng để chuyển 1 chuỗi về dạng in thường:

s = "CODELEARN123"

print(s.lower())

Kết quả khi chạy chương trình:codelearn123

### 12. Phương thức upper()

Đây là phương thức được dùng để chuyển 1 chuỗi về dạng in hoa:

s = "codelearn123"

print(s.upper())

Kết quả khi chạy chương trình:CODELEARN123

### 13. Phương thức isalnum()

Đây là phương thức được dùng để kiểm tra xem một xâu có chỉ chứa các ký tự chữ và số hay không. Ví dụ:

s = "codelearn2020"

print(s.isalnum())

s = "codelearn2020.io"

# Kết quả sẽ là False do chuỗi s chứa ký tự .

print(s.isalnum())

Kết quả khi chạy chương trình:

True

False

### 14. Phương thức isalpha()

Phương thức này được dùng để kiểm tra xem một chuỗi có chứa toàn các ký tự chữ không:

s = "codelearn"

print(s.isalpha())

# Kết quả sẽ là False do chuỗi s chứa số 2020

s = "codelearn2020"

print(s.isalpha())

Kết quả khi chạy chương trình:

True

False

### 15. Phương thức isnumeric()

Phương thức này dùng để kiểm tra xem một xâu có chứa toàn các ký tự số hay không:

s = "2020"

print(s.isnumeric())

s = "c2020"

print(s.isnumeric())

Kết quả khi chạy chương trình:

True

False

### 16. Phương thức split()

Phương thức này được dùng để cắt một chuỗi ra thành list các chuỗi khác dựa trên một phần tử trong chuỗi đầu vào:

s = "Welcome to Codelearn.io!"

print(s.split(" "))

s = "A1B1C1D1E1"

print(s.split("1"))

Kết quả khi chạy chương trình:

['Welcome', 'to', 'Codelearn.io!']

['A', 'B', 'C', 'D', 'E', '']

### 17. Phương thức join()

Phương thức này được dùng để nối một tập hợp thành một chuỗi sử dụng kí tự cho trước. Ví dụ:

lst = ["Welcome", "to", "Codelearn.io!"]

print(" ".join(lst))

lst = ["A", "B", "C"]

print("-".join(lst))

Kết quả khi chạy chương trình:

Welcome to Codelearn.io!

A-B-C

Bạn có thể sử dụng hàm split() và hàm join() để loại bỏ các khoảng trắng thừa trong chuỗi. Ví dụ:

message = " Welcome to Codelearn.io! "

print(" ".join(message.split()))

Kết quả khi chạy chương trình:

Welcome to Codelearn.io!

### 18. Phương thức replace()

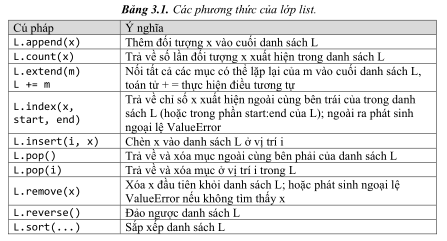
Phương thức này được dùng để thay thế các chuỗi con tìm thấy thành chuỗi con mới. Ví dụ:

name = "Cod3l3arn"

print(name.replace("3", "e"))

Kết quả khi chạy chương trình:Codelearn

# Hàm và kiểu dữ liệu kết hợp



Trong các chương trước bạn đã biết cách sử dụng các hàm có sẵn như hàm len(), upper(), lower(), sort(), ... qua bài này bạn sẽ hiểu được cách tạo và sử dụng hàm.

Hàm được hiểu đơn giản là 1 đoạn code, khi bạn gọi hàm thì thực ra là bạn đang gọi tới đoạn code này (bạn cũng có thể hiểu hàm là viết tắt của 1 đoạn code). Để khai báo một hàm bạn dùng từ khóa def giống như sau:

## def <tên\_hàm>:

# Các đoạn code trong thân hàm

Ví dụ về hàm hiển thị ra màn hình các số từ 1 tới 10:

def show():

for i in range(1, 11):

## # end = " " có nghĩa là sử dụng hàm print và không xuống dòng

print(i, end=" ")

print()

# gọi tới hàm show

show()

show()

Kết quả khi chạy chương trình:

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10

Có thể thấy cách tạo và sử dụng hàm rất đơn giản, chương trình trên hiển thị ra 2 lần các số từ 1 tới 10 do gọi tới hàm show() 2 lần.

Ngoài ra bạn cũng có thể truyền các tham số vào hàm và trả về kết quả bằng từ khóa return <giá\_trị\_trả\_về>. Ví dụ về hàm trả về tổng của một list:

## def sum\_of\_list(lst):

answer = 0

for v in lst:

answer += v

return answer

print(sum\_of\_list([3, 4, 2]))

print(sum\_of\_list([8, 4, 7]))

print(sum\_of\_list([1, 2, 3]))

Kết quả khi chạy chương trình:

9 19 6

## Ví dụ khác về hàm trả về số các số chẵn trong list:

def count\_even(lst):

count = 0

for v in lst:

if v % 2 == 0:

count += 1

return count

print(count\_even([3, 4, 2]))

print(count\_even([2, 4, 6]))

print(count\_even([7, 3, 2]))

Kết quả khi chạy chương trình:

2 3 1

# Vòng lặp và điều kiện

*Các phép toán*

- Phép toán số học: +, -, \*, /, // (chia lấy phần nguyên)

- Phép toán logic: and, or, not

- Phép gán: Biến = <Biểu thức>

*Cấu trúc rẽ nhánh if*

a) Cấu trúc rẽ nhánh if khuyết

if (btđk ):

Công\_việc

b) Cấu trúc rẽ nhánh if…else

if (btđk):

công\_việc1 khi btđk đúng;

else:

công\_việc2 khi btđk sai;

c) Cấu trúc rẽ nhánh if…elif..else

if (btđk1):

công\_việc\_1

elif(btđk2):

công\_việc\_2

…

elif(btđkn):

công\_việc\_n

else:

công\_việc\_n1

Cấu trúc if được sử dụng trong các trường hợp có sự rẽ nhánh trong tiến trình thực hiện công việc, tuỳ thuộc vào từng tình huống cụ thể của bài toán mà lựa chọn loại cấu trúc rẽ nhánh thích hợp.

**Cấu trúc lặp**

*a) Cấu trúc lặp while*

while(btđk):

Công\_việc\_LẶP;

Cấu trúc lặp while được sử dụng trong các trường hợp cần giải quyết công việc lặp đi lặp lại nhiều lần.

*b) Cấu trúc lặp for*

for<item> in <collection>:

Công\_việc\_LẶP

Cấu trúc lặp for được sử dụng trong các trường hợp giải quyết công việc lặp đi lặp lại nhiều lần, đặc biệt thường dùng khi cần duyệt các kiểu dữ liệu danh sách