Ngôn ngữ lập trình Python

Giảng viên: Phạm Đình Phong

Email: phongpd@utc.edu.vn

ĐT: 0972481813

Giới thiệu

- Tại sao Python?
  - Hỗ trợ nhiều mô hình lập trình
  - Chạy trên nhiều hệ điều hành
  - Có tính đa dạng về ứng dụng
  - Thư viện phong phú
  - Cộng đồng đông đảo

Giới thiệu

- Python?
  - Là ngôn ngữ thông dịch
  - Phiên bản hiện tại là 3.12.5
  - Website: python.org

1. Hello world

Tạo một file có tên là helloworld.py có nội dung:

```
print (‘Hello world!’)
```

Lệnh print là lệnh xuất ra màn hình

Lưu lại, vào cửa sổ command line và chuyển đến thư mục chứa file này rồi gõ lệnh:

```
python helloworld.py
```

2. Cú pháp

- Khai báo biến:

```
x = 10
```

- Có thể gán nhiều loại giá trị cho một biến

```
a = 1
a = 'Hello World'
a = [1, 2, 3]
a = [1.2, 'Hello', 'W', 2]
```

Lưu ý: không thể sử dụng biến nếu biến chưa được gán giá trị và biến không có giá trị mặc định

3. Các toán tử

- Các toán tử số học giống như các ngôn ngữ lập trình khác, ngoài ra:
  - Dùng // để chia nguyên
  - Dùng ** để tính lũy thừa
- Các toán tử logic
  - Phép tuyển: or
  - Phép hội: and
  - Phủ định: not
- Các toán tử so sánh: giống C/C++
  - >, >=, <, <=, ==, !-

3. Các toán tử

- Hỗ trợ so sánh kép:
  - x = 2
  - 1 < x < 3 # True
  - 20 < x < 10 # False
  - 3 > x <= 2 # True
  - 2 == x < 4 # True
- Toán tử tập hợp:
  - in, not in

3. Các toán tử

- Phép gán phù thủy:

```
x, y, z = 1, 2, 3
print (x, y, z)
```

- Hoán đổi giá trị của hai biến:

```
x, y = y, x
print(x, y, z)
```

3. Các toán tử

- Sequence unpacking

```
values = 1, 2, 3
print (values)
a, b, c = values
print (a)

x, y, z = 1, 2 #Báo lỗi do không có giá trị nào cho z
a, b, *rest = [1, 2, 3, 4] # Số thừa được gán cho rest
```

3. Các toán tử

- Chuỗi gán (Chained Assignments)

```
x = y = somefunction()
#Tương đương với
y = somefunction()
x = y
#Khác với
x = somefunction()
y = somefunction()
```

4. Lấy giá trị nhập từ bàn phím

x = input("Giá trị của x: ")

print (x)

Lưu ý: giá trị của x là kiểu chuỗi

y = input("Giá trị của y: ")

print (x+y)

print (int(x)+int(y))

→ So sánh sự khác nhau

5. Cấu trúc điều khiển

- Khối lệnh: được xác định dựa vào thụt đầu dòng (indent)
- Cấu trúc điều khiển if:

```
if condition1 :
    <Khối lệnh 1>
elif condition2 :
    <Khối lệnh 2>
elif condition3 :
    <Khối lệnh 3>
else:
    <Khối lệnh 4>
```

5. Cấu trúc điều khiển

- Ví dụ 1: kiểm tra số chẵn, lẻ

```
x = input("Nhap vao mot so:")
x = int(x)
if x%2 == 0:
    print(str(x) + " la so chan")
else:
    print(str(x) + " la so le")
```

5. Cấu trúc điều khiển

- Ví dụ 2: kiểm tra âm, dương

```
num = int(input('Enter a number: '))
if num > 0:
    print('The number is positive')
elif num < 0:
    print('The number is negative')
else:
    print('The number is zero')
```

5. Cấu trúc điều khiển

- if ở thể ngắn: Nếu trong khối lệnh if chỉ có một lệnh duy nhất thì có thể để lệnh đó cùng dòng.

```
if a > b: print("a is greater than b")
```

- if ... else ở thể ngắn:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

5. Cấu trúc điều khiển

- if ... else ở thể ngắn với 3 điều kiện:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

5. Cấu trúc điều khiển

- Cấu trúc: for...in

```
for iterating_var in sequence:
    <Khối lệnh>
```

- Ví dụ:

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for number in numbers:
    print ('Current number:', number)

fruits = ['banana', 'apple', 'mango']
for fruit in fruits:
    print ('Current fruit :', fruit)
```

5. Cấu trúc điều khiển

- else in for loop

  Từ khóa else trong vòng lặp for xác định một khối lệnh được thực hiện khi vòng lặp kết thúc

- Ví dụ:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

→ In các số từ 0 đến 5 và một thông báo khi vòng lặp kết thúc

5. Cấu trúc điều khiển

- Chú ý: khối else không được thực thi khi vòng lặp được kết thúc bằng lệnh break

```
for x in range(6):
    if x == 3: break;
    print(x)
else:
    print("Finally finished!")
```

→ Kết thúc vòng lặp khi x = 3.

5. Cấu trúc điều khiển

- Lặp song song

```
names = ['anne', 'beth', 'george', 'damon']
ages = [12, 45, 32, 102]
for i in range(len(names)):
    print (names[i], 'is', ages[i], 'years old')
```

- Sử dụng hàm zip()

```
for name, age in zip(names, ages):
    print(name, 'is', age, 'years old')
```

Hàm zip() trả lại chuỗi các tuples

5. Cấu trúc điều khiển

- Cấu trúc: while

```
while expression:
    <Khối lệnh>
```

- Ví dụ:

```
count = 0
while (count < 9):
    print ('The count is:', count)
    count = count + 1
```

6. Hàm

- Cú pháp

```
def function_name(param1, param2, …):
    <Khối lệnh>
```

- Hàm không trả về dữ liệu thì mặc định trả về giá trị None
- Ví dụ:

```
def sum(a, b):
    return (a+b)
```

- Gọi hàm

```
sum(1, 2)
```

6. Hàm

- Hàm có đối mặc định

```
def plus(c, d = 10):
    return (c+d)
```

- Gọi hàm

```
plus(2)
plus(2, 3)
```

6. Hàm

- Thay đổi thứ tự tham số truyền vào

```
def sum(a, b):
    return (a+b)
```

- Gọi hàm

```
sum(b = 1, a = 10)
```

7. Xử lý chuỗi/xâu

- Một chuỗi có thể nằm trong dấu nháy đơn hoặc kép

  str1 = "Hello"

  str2 = 'world'

- Có thể truy cập từng ký tự trong chuỗi qua chỉ số. Ví dụ: str1[0], str1[1], …

7. Xử lý chuỗi/xâu

- Có thể sử dụng 3 dấu nháy (kép hoặc đơn) để khai báo chuỗi trên nhiều dòng
- Ví dụ:

```
p = """This is line 1
This is line 2
This is line 3"""
```

7. Xử lý chuỗi/xâu

- Nối chuỗi

```
str = str1 + " " + str2
```

- Lấy xâu con sử dụng [start:end]. Mặc định start là từ vị trí đầu chuỗi (0) và end là đến vị trí cuối chuỗi.
- Ví dụ:

```
str = 'Hello world'
print (str[0:4])
print (str[:4])
print (str[-3:])
```

7. Xử lý chuỗi/xâu

- Lấy độ dài chuỗi sử dụng hàm len()

```
count = len("Hello world")
```

- Tìm và thay thế chuỗi

```
replace(search, replace[, max])
```

- Ví dụ:

```
str = 'Hello world'
newstr = str.replace('Hello', 'Bye')
print (newstr)
```

7. Xử lý chuỗi/xâu

- Tìm vị trí chuỗi con

  find(str, beg=0, end=len(string))

  Hàm find tìm từ vị trí 0, nếu không tìm thấy trả về -1

- Ví dụ:

```
str = 'Hello world'
print (str.find('world'))
print (str.find('Bye'))
```

- Để tìm từ phải sang trái, sử dụng hàm rfind()

7. Xử lý chuỗi/xâu

- Tách chuỗi

  split(str=" ", num=string.count(str))

- Ví dụ:

  str = 'Hello world'

  print (str.split(' '))

  Trả về một mảng có hai phần tử là hai chuỗi "Hello" và "world"

- Sử dụng hàm splitlines() để tách chuỗi theo từng hàng và loại bỏ ký tự xuống dòng (Newline)

7. Xử lý chuỗi/xâu

- Một số hàm/phương thức xử lý chuỗi
  - Loại bỏ ký tự trắng: strip(), lstrip(), rstrip()
  - isnumeric() : Kiểm tra một chuỗi có phải là chuỗi số
  - lower() : Chuyển toàn bộ chuỗi thành chữ thường
  - upper() : Chuyển toàn bộ chuỗi thành chữ HOA

7. Xử lý chuỗi/xâu

- Một số lưu ý khi sử dụng xâu

  str = '"Hello, world!" she said'

  print (str)

  → "Hello, world!" she said

  str = 'let's go!'

  → SyntaxError: invalid syntax

  str = 'Let\'s go!'

  → Let's go!

  str = "\"Hello, world!\" she said"

  → "Hello, world!" she said

7. Xử lý chuỗi/xâu

- Chuỗi thô (raw string)

```
print('Hello,\nworld!')
```

Kết quả:

```
Hello,
world!
```

```
path = 'C:\nowhere'
print (path)
```

Kết quả:

```
C:
owhere
```

7. Xử lý chuỗi/xâu

- Chuỗi thô (raw string)

```
path = 'C:\\nowhere'
print (path)
```

Kết quả:

```
C:\nowhere
```

```
path = r'C:\nowhere'
print (path)
```

8. Danh sách (List)

- Danh sách List là cấu trúc tuần tự, các phần tử trong List có thể không cùng kiểu
- Khai báo:

```
edward = ['Edward Gumby', 42]
john = ['John Smith', 50]
```

- Một List có thể chứa các List khác

```
cl = [edward, john]
print (cl)
```

8. Danh sách (List)

- Các thao tác trên List
  - Truy cập phần tử thông qua chỉ số, phần tử đầu tiên có chỉ số là 0
  - Ví dụ 1:

```
g = 'Hello'
print (g[0])
print (g[-1])
```

  - Ví dụ 2:

```
numbers = [1, 2, 3, 4, 5]
names = ['Marry', 'Peter']
print (str(numbers[0]) + '. ' + names[1])
```

8. Danh sách (List)

- Trích xuất dãy con: tương tự như chuỗi

```
alphabet = ['a', 'b', 'c', 'd']
print (alphabet[:2])
print (alphabet[:-2])
```

Trích xuất với bước nhảy dài (mặc định bước nhảy là 1)

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print (numbers[0:10:2])
print (numbers[0:10:3])
```

8. Danh sách (List)

- Xóa phần tử: dùng del

```
numbers = [1, 2, 3, 4, 5]
del numbers[0]
print (numbers)
```

- Xóa một khoảng

```
numbers = [1, 2, 3, 4, 5, 6, 7]
del numbers[2:4]
print (numbers)
```

8. Danh sách (List)

- Nối danh sách: dùng toán tử +

```
a = [1, 2]
b = [1, 3]
c = a + b
print (c)
```

- Thêm phần tử: dùng phương tức bằng append

```
numbers = [1, 2, 3]
numbers.append(4)
print (numbers)
```

8. Danh sách (List)

- Lấy phần tử cuối danh sách bằng pop

```
numbers = [1, 2, 3]
mynum = numbers.pop() #lấy giá trị và xóa phần tử cuối
print (mynum)
print (numbers)
```

- Tìm một giá trị bằng index

```
aList = [123, 'xyz', 'zara', 'abc'];
print ("Index for xyz : ", aList.index('xyz'))
print ("Index for zara : ", aList.index('zara'))
```

8. Danh sách (List)

- Đảo ngược bằng reverse

```
numbers = [1, 2, 3, 4]
numbers.reverse()
print (numbers)
```

- Sắp xếp giá trị các phần tử bằng sort

```
x = [4, 6, 2, 1, 7, 9]
x.sort()
print ("Accending List: ", x)

x = [4, 6, 2, 1, 7, 9]
x.sort(reverse=True)
```

8. Danh sách (List)

- Lấy độ dài của List bằng len

```
numbers = [100, 34, 678]
print (len(numbers))
```

- Lấy phần tử lớn nhất và nhỏ

```
numbers = [100, 34, 678]
print (max(numbers))
print (min(numbers))
```

8. Danh sách (List)

- Phương thức copy

  Với lệnh gán thì biến mới chỉ tham chiếu tới biến gốc

```
a = [1, 2, 3]
b = a
b[1] = 4
print (a)
```

  → kết quả là: 1, 4, 3

8. Danh sách (List)

- Phương thức copy

  Muốn a và b là các danh sách độc lập thì dùng copy

```
a = [1, 2, 3]
b = a.copy()
b[1] = 4
print (a)
print (b)
```

→ kết quả là:

1, 2, 3

1, 4, 3

8. Danh sách (List)

- Xem thêm các phương thức
  - count: đếm số lần xuất hiện của một phần tử (số, chuỗi, danh sách) trong một danh sách
  - extend: mở rộng danh sách
  - insert: chèn phần tử vào danh sách
  - remove: xóa phần tử xuất hiện đầu tiên

9. Tuple

- Tương tự cấu trúc List nhưng không thay đổi được giá trị (không có các phương thức thay đổi dữ liệu như append(), pop(), …)
  - Khai báo sử dụng dấu ()

```
mytuple = ('x', 'y', 'z')
print (mytuple)

t = tuple([1, 2, 3]) #chuyển đổi List sang Tuple
a = tuple('abc') #chuyển một xâu sang Tuple
t = tuple(3 * (40 + 2,)) #chuyển biểu thức sang Tuple
```

10. Bài tập

Bài 1: Viết chương trình nhập vào một dãy số. In dãy số ra màn hình. Tính tổng các số chẵn.

Bài 2: Viết chương trình in ra các số của n!

Bài 3: Viết chương trình tính lũy thừa (x mũ n).

Bài 4: Viết chương trình in dãy số Fibonacci.

11. Phân chia module

- Cần chia file lớn thành các module để dễ bảo trì và tái sử dụng
- Có 3 loại module thường thấy là:
  - Viết bằng Python: có phần mở rộng là .py
  - Các thư viện liên kết động: có phần mở rộng là .dll, .pyd ,.so ,.sl, ...
  - C-Module liên kết với trình biên dịch

11. Phân chia module

- Tải một module bằng lệnh

  import <tên module>

- Với lệnh trên, trình dịch sẽ tìm file chứa module theo thứ tự:
  - Thư mục hiện hành mà file nguồn đang gọi
  - Các thư mục trong PYTHONPATH
  - Các thư mục cài đặt chuẩn trên Windows/Linux/Unix

11. Phân chia module

- Khai báo một module
  - Một file python mymath.py có nội dung:

```
def cong(a, b):
        return a + b
def tru(a, b):
        return a - b
def nhan(a, b):
        return a * b
```

11. Phân chia module

- Khai báo một module
  - Tạo một file có tên myexample.py trong cùng thư mục với file mymath.py có nội dung:

```
import mymath
num1 = 1
num2 = 2
print ('Tong hai so la: ', mymath.cong(num1, num2))
```

Hoặc:

```
from mymath import cong, tru, nhan
```

Hoặc

```
from mymath import *
```

11. Phân chia module

- Gói (package) module
  - Có thể gom nhiều module .py vào một thư mục với tên thư mục là tên package
  - Tạo một file __init__.py trong thư mục này. cấu trúc thư của một package sẽ như sau:

```
|-- mypack
| |-- __init__.py
| |-- mymodule1.py
| |-- mymodule2.py
|
```

11. Phân chia module

- Gói (package) module
  - Sử dụng mymodule1 theo cú pháp import sau:

    import mypack.mymodule1
  - Hoặc

    import mypack.mymodule1 as mymodule1
  - Hoặc

    import mypack.mymodule1 as mod

11. Phân chia module

- Gói (package) module
  - Khi sử dụng một module thuộc một package thì các lệnh trong file __init__.py sẽ được thực hiện trước
  - File __init__.py bắt buộc phải có cho dù là file rỗng
  - Có thể tạo các gói con (subpackage) bên trong một package theo đúng cấu trúc thư mục, có file __init__.py
  - Ví dụ:

```
import mypack.mysubpack.mysubsubpack.module
```

12. Lớp

- Khai báo một lớp

```
class myclass([parentclass]):
        assignments
        def __init__(self):
                statements
        def method1(self):
                statements
        def method2(self):
                statements
```

12. Lớp

- Khai báo một lớp

```
class Animal():
        name = ''
        age = 0
        def __init__(self, name = '', age = 0):
                self.name = name
                self.age = age
        def show(self):
                print ('My name is ', self.name)
        def run(self):
                print ('Animal is running...')
        def go(self):
                print ('Animal is going...')
```

12. Lớp

- Khai báo một lớp

```
class Dog(Animal):
        def run(self):
                print ('Dog is running...') #Nạp đè hàm này của Animal

myanimal = Animal()
myanimal.show()
myanimal.run()
myanimal.go()
mydog = Dog('Lucy')
mydog.show()
mydog.run()
mydog.go()
```

12. Lớp

- Mặc định các phương thức có thể truy cập được từ bên ngoài.
- Thêm __ vào trước tên phương thức để biến nó thành private

```
class Secretive:
    def __inaccessible(self):
        print("Bet you can't see me ...")
    def accessible(self):
        print("The secret message is:")
        self.__inaccessible() #Vẫn truy cập được

s = Secretive()
s.__inaccessible() #Không truy cập được
```

13. Bài tập

- Sử dụng List để:
  - Viết lớp Stack bao gồm các thao tác: thêm (push), lấy phần tử (pop), lấy kích thước (size), kiểm tra rỗng (isEmpty), lấy giá trị ở đỉnh ngăn xếp (top)
  - Viết lớp Queue bao gồm các thao tác: thêm (enqueue), lấy phần tử (dequeue), lấy kích thước (size), kiểm tra rỗng (isEmpty) , lấy giá trị (front - không xóa phần tử)

14. Set (tập hợp)

- Tập không được sắp các phần tử, trong đó mỗi phần tử là duy nhất và thay đổi được
- Tạo một tập hợp bằng hàm set() hoặc {...}

```
my_set = {1, 2, 3}
print (my_set)
```

- Có thể trộn các kiểu dữ liệu

```
my_set = set(1.0, "Hello", (1, 2, 3))
print (my_set)
```

- Tạo từ danh sách

```
my_set = set([1, 2, 3, 4])
print(my_set)
```

14. Set (tập hợp)

- Không thể truy cập các phần tử của set bằng chỉ số hoặc slicing (lấy khoảng)
- Thêm phần tử

```
my_set = {1, 3}
print(my_set)
my_set.add(2)
print(my_set)
```

- Thêm nhiều phần tử

```
my_set.update([2, 3, 4]) #các phần tử trùng bị loại bỏ
print(my_set)
```

14. Set (tập hợp)

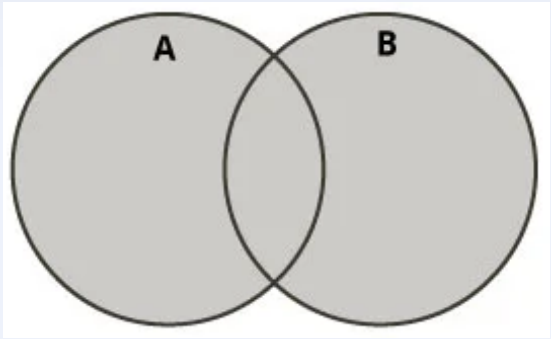
- Xóa phần tử sử dụng hàm discard() và remove(). Điểm khác là remove() báo lỗi nếu không tồn tại phần tử cần xóa.

```
my_set = {1, 3, 4, 5, 6}
print (my_set)
my_set.discard(4)
print (my_set))
my_set.remove(6)
print(my_set)
```

14. Set (tập hợp)

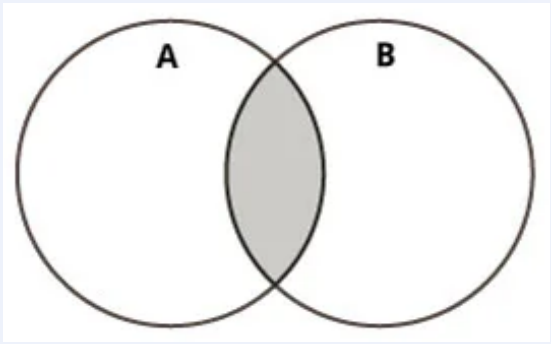
- Các phép toán trên tập hợp

Phép hợp sử dụng | hoặc union()




```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
print (A.union(B))
print (B.union(A))
```

Phép giao sử dụng & hoặc intersection()



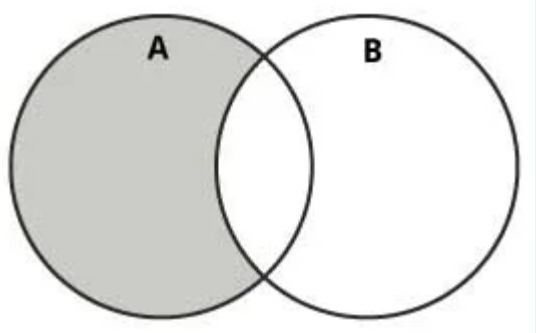

```
print (A.intersection(B))
print (B.intersection(A))
```

14. Set (tập hợp)

- Các phép toán trên tập hợp

  Phép trừ sử dụng – hoặc difference()

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
print(A - B)
print(A.difference(B))
print(B - A)
print(B.difference(A))
```



- Duyệt các phần tử

```
for letter in set("apple"):
    print(letter)
```

15. Hàng đợi

- Hàng đợi hai đầu

```
from collections import deque
q = deque(range(5))
q.append(5)
q.appendleft(6)
print (q)
print (q.pop())
print (q.popleft())
q.rotate(3)
print (q)
```

16. Dictionary

- Danh sách các phần tử
  - Mỗi phần tử gồm khóa (key) và giá trị (value)
  - Các phần tử được sắp theo giá trị khóa (tùy version)
- Khai báo bằng dấu {…} và truy xuất phần tử dựa vào khóa

```
phonebook = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
print (phonebook['Alice'])
```

16. Dictionary

- Tạo từ điển sử dụng hàm dict

```
items = [('name', 'Gumby'), ('age', 42)]
d = dict(items)
print (d)
print (d['name'])

d = dict(name='Gumby', age=42)
print (d)
```

→ kết quả: {'age': 42, 'name': 'Gumby'}

16. Dictionary

- Các phương thức
  - clear(): xóa tất cả các phần tử của tử điển
  - copy(): sao chép phần tử nhưng vẫn tham chiếu đến cấu trúc ở từ điển nguồn.
  - deepcopy(): sao chép cả các phần tử và cấu trúc
  - items(): lấy các phần tử trong từ điển ra một danh sách và mỗi phần tử tạo thành (key, value)
  - pop(): lấy phần tử có giá trị key và xóa phần tử này khỏi từ điển
  - popitem(): xóa phần tử cuối từ điển
  - update():
  - has_key(key)

17. OrderedDict

- OrderedDict ghi nhớ thứ tự của phần tử lúc chèn vào từ điển lần đầu tiên

```
from collections import OrderedDict
print("Before:\n")
od = OrderedDict()
od['a'] = 1
od['b'] = 2
od['c'] = 3
od['d'] = 4
for key, value in od.items():
    print(key, value)
print("\nAfter:\n")
od['c'] = 5
for key, value in od.items():
    print(key, value)
```

18. Heapq – Thuật toán heap queue

- heapq cung cấp một cài đặt của thuật toán hàng đợi ưu tiên

  from heapq import *

  heapify() → chuyển một List thành một heap

  heappush(heap, item) → thêm item vào heap

  heappop(heap) → xóa và trả lại phần tử nhỏ nhất

  heappushpop() → them item vào heap và xóa phần tử nhỏ nhất

  heapreplace(heap, item) → lấy và trả lại phần tử nhỏ nhất trong heap đồng thời thêm mới item

Hết