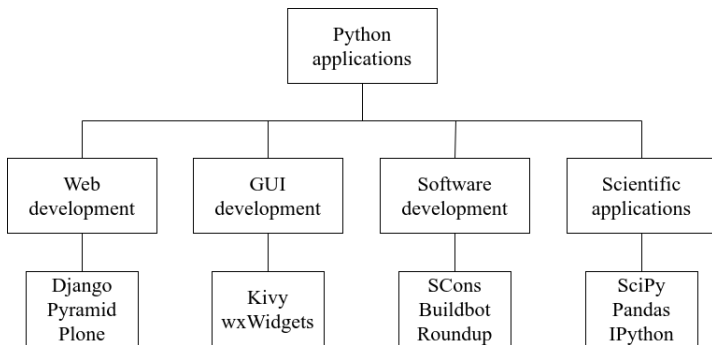


Giới thiệu Python

Một số khái niệm cơ bản

HK2, Năm học 2019 - 2020

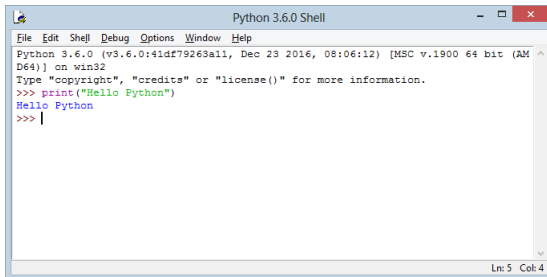
Ứng dụng của Python



<https://www.python.org/about/apps/>

Cài đặt Python

- Tải gói cài đặt tại <https://www.python.org/downloads/>
- Cài đặt
- Sử dụng *Python Shell* (chạy IDLE(Python x.x xbit), IDE tích hợp sẵn của Python)



```
Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello Python")
Hello Python
>>> |
```

- Để sử dụng dễ hơn, cài IDE như PyCharm, Eclipse
<https://www.jetbrains.com/pycharm/> (PyCharm)

Cài đặt Anaconda

- Nền tảng mã nguồn mở trên Python
- Yêu cầu: Win 7/8/10, Ubuntu 12.04+...; RAM tối thiểu 4GB; ổ cứng trống tối thiểu 3GB
- Tải cài đặt tại: <https://www.anaconda.com/>
- Lưu ý: chọn tùy chọn *"Add Anaconda to my PATH environment variable"*
- Mở Anaconda command line để cài library

```
conda install -c anaconda numpy
conda install -c anaconda pandas
conda install -c anaconda matplotlib
```
- Cài IDE mong muốn hoặc sử dụng **Spyder** tích hợp sẵn trong Anaconda

Đoạn code đơn giản

```
1 x = 14 - 3
2 y = "Hello"
3 z = 3.45
4 if (z == 3.45) or (y == "Hello"):
5     x = x + 1
6     y = y + " World"
7 print(x)
8 print(y)
```

Cú pháp

- Khoảng cách đầu dòng có ý nghĩa rất quan trọng; thật lẽ để chỉ ra một khối (block) mã
- Comment: sử dụng **#** cho 1 dòng và **"""**

```
1 # one line
2 print("Hello, World!")
3 """
4 block
5 """
6 print("Hello, World!")
```

Biến

- Không có câu lệnh khai báo biến
- Biến được tạo khi lần đầu tiên được gán giá trị; không cần khai báo kiểu và có thể thay đổi kiểu sau khi gán trị

```
1 x = 3.5
2 y = "abc"
3 x = "efg"
4 y = 123
```

- Tên biến: bắt đầu là chữ hoặc `_`, không bắt đầu với chữ số, chỉ sử dụng chữ/số (A-z, 0-9) và `_` cho tên, có phân biệt hoa - thường

Kiểu dữ liệu

- Một số kiểu dữ liệu cơ bản

Text: str

Numeric: int, float, complex

Sequence: list, tuple, range

Mapping: dict

Set: set, frozenset

Boolean: bool

Binary: bytes, bytearray, memoryview

- Kiểm tra kiểu bằng **type(x)**

Kiểu dữ liệu

Ví dụ:

```
1 x = "Hello World!" #str
2 x = 20 # int
3 x = 20.5 # float
4 x = ["a","b","c"] #list
5 x = ("a","b","c") # tuple
6 x = range(4) # range
7 x = {"name": "An", "age": 22} # dict
8 x = {"a","b","c"} # set
9 x = True # bool
```

Kiểu số - Ép kiểu

```
1 # x = 1
2 x = int(1)
3 x = int(1.7)
4 x = int("1")
5 # ham float
6 x = float(1)      # 1.0
7 x = float(1.7)    # 1.7
8 x = float("1.7")  # 1.7
9
10 x = str("s1")     # x = 's1'
11 y = str(2)        # y = '2'
12 z = str(3.0)      # z = '3.0'
```

Kiểu chuỗi

- 'str' và "str" là như nhau
- print(): hiển thị chuỗi ra màn hình

```
1 a = "Hello World"
2 print(a)
3 b = """Dong 1
4     Dong 2
5     Dong 3 """ # Nhiều dòng
6 b = ''' Dong 1
7     Dong 2
8     Dong 3 ''' # Nhiều dòng
9
10 print(a[1])           # "e"
11 print(a[2:4])         # "ll"
12 print(a[-5:-2])       # "orl"
13 print(len(a))         # 13
```

Kiểu chuỗi

```
1 x = "    Hello, World!    "  
2 a = x.strip()                # "Hello, World!"  
3 print(a.lower())             # "hello, world!"  
4 print(a.upper())             # "HELLO, WORLD!"  
5 print(a.replace("H","J"))    # "Jello, World!"  
6 print(a.split(","))          # ["Hello", " World!"]  
7 b = "ear" not in a           # True  
8 b = "Hallo" + ", " + "Welt!" # 'Hallo, Welt!'  
9 c = "I am " + 22              # Error  
10 txt = "Toi {} va chi toi {}"  
11 print(txt.format(22, 24))    # Toi 22 va chi toi 24  
12 txt = "Tong tien {2}: bao gom gao {1}, thuc an {0}"  
13 print(txt.format(10.2, 20.8, 31)) # Ket qua?  
14 txt = "Phim dang chieu: \"Frozen\" vao 18h"
```

Kiểu List

Tập hợp có thứ tự và có thể thay đổi. Cho phép dữ liệu trùng nhau

```
1 ds = ["apple", "banana", "cherry", "fig", "grapes", "kiwi", "orange"]
2 print(ds)
3 print(ds[1])      # banana
4 print(ds[-2])     # kiwi
5 print(ds[4:])     # grapes, kiwi, orange
6 print(ds[-4:-1]) # fig, grapes, kiwi
7 ds[1] = "lemon"   # thay doi gia tri cua list
8 for x in ds:      # duyệt các phần tử
9     print(x)
10 if "apple" in ds: # kiểm tra tồn tại
11     print("Have apple")
12 print(len(ds))   # số phần tử
```

Kiểu List

```
1 ds = ["apple", "banana"]
2 ds.append("fig") # apple, banana, fig
3 ds.insert(1, "lemon") # apple, lemon, banana, fig
4 ds.remove("banana") # apple, lemon, fig
5 ds.pop() # apple, lemon
6 ds.pop(0) # lemon
7 ds = ["apple", "banana", "cherry"]
8 del ds[1] # apple, cherry
9 ds.clear # []
10 ds1 = ["a", "b", "c"]
11 ds2 = ds1.copy(); # copy mot ds vao 1 ds khac
12 ds3 = [1, 2]
13 ds4 = ds1 + ds3 # 'a', 'b', 'c', 1, 2
14 ds2.extend(ds3)
15 print(ds2) # 'a', 'b', 'c', 1, 2
16 ds5 = list(('a', 'b', 'c', 'd')) # tao ds
```

Kiểu Tuple

Tập hợp có thứ tự và **không** thể thay đổi.

```
1 tp1 = ("a", "b", "c", "d", "e")
2 print(tp1[1])    # 'b'
3 print(tp1[-2])   # 'd'
4 print(tp1[1:3])   # 'b', 'c'
5 print(tp1[-3:-1]) # 'c', 'd'
```

Khi tuple được tạo, chúng ta không thể thay đổi giá trị.
Tuy nhiên để thay đổi, chúng ta có thể thực hiện

```
1 x = ("a", "b", "c")
2 y = list(x)
3 y[1] = "d"
4 x = tuple(y)
```

Kiểu Tuple

Các cú pháp truy cập phần tử, kiểm tra tồn tại, số phần tử tương tự của `list`.

Không thể thực hiện: **thêm, thay đổi** phần tử

```
1 tpl = (1,) # tao tuple voi 01 phan tu
2 tpl1 = ('a', 'b', 'c')
3 tpl2 = (1, 2, 3)
4 tpl3 = tpl1 + tpl2 # ghép 02 tuple
5
6 tpl4 = tuple(('x', 'y', 'z')) # cach khac de tao tuple
```


Kiểu Set

Tập hợp không thứ tự, không chỉ mục

```
1 st = {"a", "b", "c"}
2 for x in st:
3     print(x)
4 st.add("d")
5 st.add(["e", "f"]) # Them nhieu phan tu
6 st.remove("a") # error neu phan tu khong ton tai
7 st.discard("a") # no error neu phan tu khong ton tai
8 st.pop()
9 st.clear()
10
11 set1 = {"a", "b", "c"}
12 set2 = {1, 2, 3}
13 set3 = set1.union(set2) # bo phan tu trung
14 set1.update(set2) # bo phan tu trung
15
16 set4 = set(("a", "b"))
```

Kiểu Dictionary

Không thứ tự, có thể thay đổi và có chỉ mục

```
1 dict = {"ho": "Nguyen", "ten": "An", "tuoi": 18}
2 x = dict["ho"]
3 x = dict.get("ho")
4 dict["ho"] = "Tran"
5 for key in dict:
6     print(key)           # In key
7     print(dict[key])     # in Value
8 for val in dict.values():
9     print(val)
10 for key, val in dict.items():
11     print(key, val)
12 dict.pop("gt")
13 dict.popitem() #
```

Toán tử

- Số học: `+`, `-`, `*`, `/`, `%`, `**`, `//`
- Gán: `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `//=`, `**=`, `...`
- So sánh: `==`, `!=`, `>`, `<`, `>=`, `<=`
- Logic: `and`, `or`, `not`
- Thành viên: `in`, `not in`

if Statements

```
1 if x == 3:
2     # do something
3 elif x == 2:
4     # do something
5 else:
6     # do something
```

Loop Statements

```
1 i = 1
2 while i < 6:
3     i += 1
4     if i == 5:
5         break
6     if i == 2:
7         continue
8     print(i)
9 for x in mylist:
10    print(x)
11 for i in range(5):
12    print(x)
13 for (i, item) in enumerate(mylist):
14    print(i, item)
```

List Comprehension

Tạo một list mới bằng cách áp dụng hàm cho mọi phần tử của danh sách ban đầu `[expression for name in list]`

```
1 li = [3, 6, 2, 7]
2 li1 = [ele * 2 for ele in li]
3 li = [('a', 1), ('b', 2), ('c', 7)]
4 li1 = [n*3 for (x, n) in li]
5 li = [3, 6, 2, 7, 1, 9]
6 li1 = [ele * 2 for ele in li if ele > 4]
```

Hàm (Function)

```
1 def myfunc(x, y):  
2     x += 1  
3     y += 2  
4     return x * y  
5 print(myfunc(3,4))  
6  
7 def apply(f, x, y):  
8     return f(x)  
9 print(apply(myfunc, 3, 4))
```

Lambda

Hàm có thể được định nghĩa không cần tên

```
1 x = lambda a: a + 10
2 print(x(5))
3
4 x = lambda a, b: a + b
5 print(x(3, 4))
```


String

String format()

```
1 maxnumber = 100
2 txt = "Max grade is {}!!!"
3 print(txt.format(maxnumber))
4 txt = "Max grade is {:.2f}"
5 print(txt.format(maxnumber))
```

Khi có nhiều biến truyền vào chuỗi:

```
1 txt = "Diem mon {}: {}"
2 print(txt.format("Co so du lieu", 8))
3 txt = "Diem mon {1}: {0}"
4 print(txt.format(8, "Co so du lieu"))
```

File

`open()` gồm 2 biến: `filename`, `mode`

Với mode có giá trị: `"r"` (để đọc), `"a"` (mở file để nối vào), `"w"` (để viết), `"x"` (tạo một file cụ thể); `"t"` (text), `"b"` (binary)

```
1 f = open("demo.txt")
2 # f = open("demo.txt", "rt")
3 print(f.read())
4 print(f.read(5)) # doc 5 ky tu
5 print(f.readline()) # doc 1 dong
6 for x in f:
7     print(x) # doc tung dong
8 f.close()
9
10 f = open("demo.txt", "a")
11 f.write("Hello world")
12 f.close()
```

Class/Object

Tạo lớp: `class`

```
1 class SinhVien:  
2     lop = 'CNTT'
```

Tạo đối tượng của lớp:

```
1 sv1 = SinhVien()  
2 print(sv1.lop) # SV
```

Hàm khởi tạo `__init__`: thực thi khi lớp được khởi tạo

```
1 class SinhVien:  
2     def __init__(self, ten, tuoi):  
3         self.ten = ten  
4         self.tuoi = tuoi  
5     sv1 = SinhVien("Bao", 22)  
6     print(sv1.ten, sv1.tuoi) # Bao, 22
```

Class/Object

Biến **self**: tham chiếu đến thể hiện hiện tại của lớp, được sử dụng để truy cập các biến thuộc lớp; không cần đặt tên **self**, có thể đặt lại tên nhưng phải là tham số đầu tiên của hàm

```
1 class SinhVien:
2     def __init__(selfvar, ten, tuoi):
3         selfvar.ten = ten
4         selfvar.tuoi = tuoi
5     def print_ten(selfvar1):
6         print(selfvar1.ten)
7     sv1 = SinhVien("Duong", 22)
8     sv1.print_ten()
```

Class/Object

Tạo lớp con (Child) thừa kế chức năng của lớp cha

```
1 class SVCNTT(SinhVien):
2     def __init__(self, ten, tuoi, khoa):
3         # SinhVien.__init__(ten, tuoi)
4         super().__init__(ten, tuoi)
5         self.khoa = khoa
6     def innd(self):
7         print(self.ten, self.tuoi, self.khoa)
8     svcntt1 = SVCNTT("Duy", 22, 7 )
9     svcntt1.innd()
```

Collections/Deque

`deque` được phát triển trong module `collections`; thường được sử dụng khi:

- cần thao tác `append` và `pop` nhanh từ cả 02 đầu
- `append()`: thêm giá trị về phía **phải** của hàng đợi
- `appendleft()`: thêm giá trị về phía **trái** của hàng đợi
- `pop()`: xóa phần tử về phía **phải**
- `popleft()`: xóa phần tử về phía **trái**
- `remove()`: xóa phần tử đầu tiên có giá trị cần tìm
- `count()`: đếm số lần xuất hiện của giá trị trong hàng đợi
- ...

```
1 from collections import deque
2 de = deque([1,2,3])
3 de.append(4) # 1,2,3,4
4 de.appendleft(0) # 0,1,2,3,4
5 de.pop() # 0,1,2,3
6 de.popleft() #1,2,3
```

Heap queue (heapq)

Heap được sử dụng để biểu diễn hàng đợi ưu tiên; thuộc module `heapq`

- `heappush(push, item)`: chèn item vào heap; thứ tự sẽ được điều chỉnh để cấu trúc heap được duy trì
- `heappop(heap)`: xóa và trả về phần tử nhỏ nhất trong heap; thứ tự trong heap cũng được điều chỉnh để cấu trúc heap được đảm bảo
- ...

```
1 import heapq
2 li = [5, 7, 9, 1, 3]
3 heapq.heapify(li)
4 heapq.heappush(li, 4)
5 print (heapq.heappop(li))
```