




1

 **Nội dung chính**

- 1. Giới thiệu ngôn ngữ Python.**
- 2. Định danh.**
- 3. Cú pháp Python căn bản.**
- 4. Cấu trúc rẽ nhánh.**
- 5. Cấu trúc lặp.**
- 6. Hàm.**
- 7. Module & Package.**
- 8. Xử lý ngoại lệ.**

Dương Hữu Thành @2021 Khoa CNTT

2



Giới thiệu ngôn ngữ Python

- Python là ngôn ngữ lập trình cấp cao, đơn giản, dễ mở rộng.
- Python cho phép chia chương trình thành các module để tái sử dụng.
- Python là một cách tiếp cận lập trình hướng đối tượng hiệu quả.
- Python là ngôn ngữ **thông dịch**.

3



Giới thiệu ngôn ngữ Python

- Python viết chương trình chặt chẽ và dễ đọc
 - Các câu lệnh được nhóm bằng indentation.
 - Các câu lệnh không cần kết thúc bằng dấu chấm phẩy (;).
 - Không cần khai báo trước biến hay đối số.

4

Cài đặt môi trường

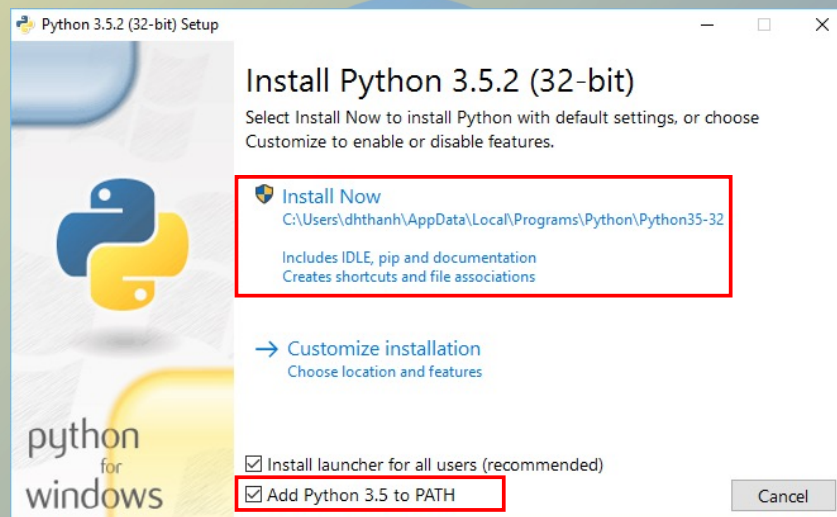
- Cài đặt Python trên Window
 - Tải Python: <https://www.python.org/downloads/>
 - Double-click vào tập tin cài đặt bình thường.



5

Cài đặt môi trường

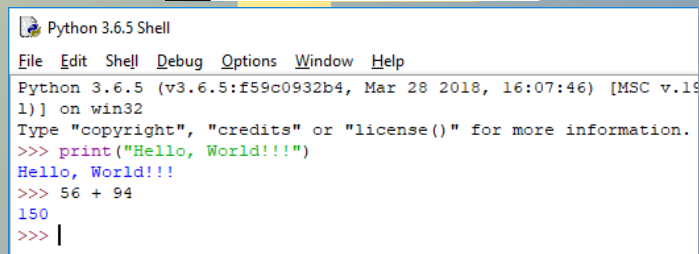
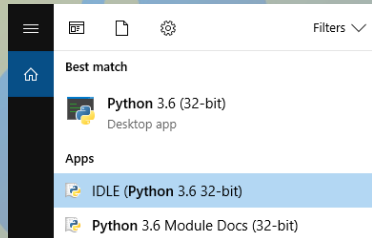
- Chú ý: trong các bước cài đặt Python nên check cập nhật python vào biến môi trường PATH.



6

Cài đặt môi trường

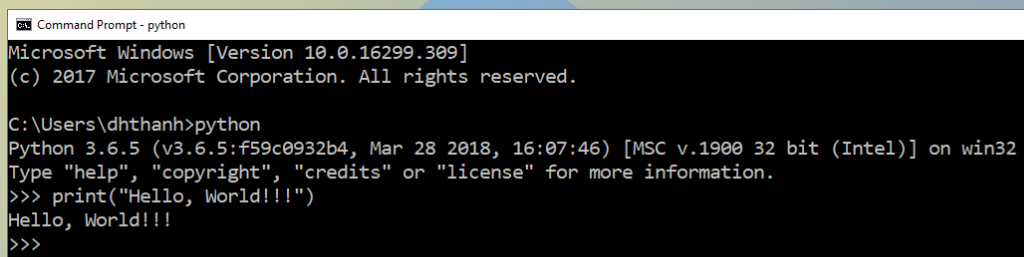
- Môi trường tiếp cận lập trình Python đơn giản nhất là Python Shell, tìm kiếm từ khóa "python" trong menu Start và mở chương trình.

A screenshot of the Python 3.6.5 Shell window. The window title is 'Python 3.6.5 Shell'. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following content:

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello, World!!!")
Hello, World!!!
>>> 56 + 94
150
>>> |
```

Cài đặt môi trường

- Hoặc đơn giản gõ lệnh python trong Command Prompt và bắt đầu viết chương trình.

A screenshot of the Windows Command Prompt window. The window title is 'Command Prompt - python'. The text area shows the following content:

```
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\dhthanh>python
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!!!")
Hello, World!!!
>>>
```




Định danh

- Định danh (identifier) là tên dùng xác định một hàm, một biến, một đối tượng, một lớp, một module trong python.
- Python là ngôn ngữ phân biệt hoa, thường (case sensitive), tức là hai định danh `my_var` và `my_VAR` là khác nhau.



Một số quy ước đặt tên định danh

- Tên định danh trong Python phải bắt đầu bằng các ký tự $a \rightarrow z$ hoặc $A \rightarrow Z$, hoặc các ký tự gạch chân `_`.
- Ký tự số không được bắt đầu cho tên định danh.
- Không sử dụng ký tự đặc biệt trong tên định danh như `@`, `#`, `$`, `%`, `^`, v.v.
- Các từ khóa không được dùng làm tên định danh.
- Tên lớp (class) phải bắt đầu bằng ký tự hoa, còn tên các định danh khác (biến, hàm, đối tượng, module) bắt đầu bằng ký tự thường.



Cú pháp căn bản

- Python không sử dụng cặp dấu ngoặc nhọn {} để để chỉ các khối mã nguồn (block of codes), mà sử dụng indentation (thụt dòng) để làm điều đó.
- Số indetation của mỗi dòng lệnh có thể thay đổi, nhưng các lệnh trong cùng khối phải cùng số indentation.
- Các dòng tiếp theo nhau thì số lượng indentation thụt vào phải giống nhau.

11



Cú pháp căn bản

- Ví dụ

```
if True:
    print("Great!")
else:
    print("Sorry!")
```

đúng

```
if True:
    print("Great!")
else:
    print("Sorry!")

print("Thanks")
```

sai

```
if True:
    print("Great!")
else:
    print("Sorry!")

print("Thanks")
```

đúng

12



Cú pháp căn bản

- Viết lệnh trên nhiều dòng
 - Các lệnh trong python kết thúc bằng newline. Tuy nhiên, ta có thể sử dụng \ thể hiện sự nối tiếp để viết lệnh trên nhiều dòng.
 - Các lệnh bên trong dấu [], {}, () thì không cần sử dụng \ khi viết trên nhiều dòng.

```
x = a + b + \
    d + e + \
    f

a1 = ["a", "b",
      "c", "d"]
```



Cú pháp căn bản

- Ta cũng có thể viết nhiều lệnh trong cùng dòng và sử dụng dấu chấm phẩy (;) để cách các lệnh.

```
a = 5; b = 10
if a < b: print("%s < %s" % (a, b)); print("Next Cmd")
```



Ghi chú

- Ghi chú (comment) trong Python sử dụng:
 - # comment trên một dòng
 - Trong cặp dấu nháy ba `"""` `"""` để viết comment trên nhiều dòng

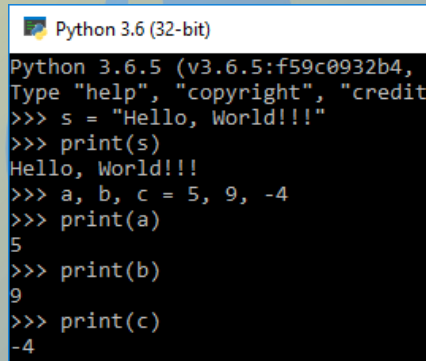
```
# Nhập x
x = int(input("x = "))
"""
    Nếu x > 0 thì in ra màn hình "x > 0"
    Ngược lại in ra màn hình "x <= 0"
"""
if x > 0:
    print("x > 0") # in kết quả x > 0
else:
    print("x <= 0") # in kết quả x <= 0
```



Các phép toán số học

- Phép toán cộng (+), trừ (-), nhân (*), chia (/) và sử dụng dấu () để nhóm các phép toán.
- Phép chia (/) luôn trả về số thực.
- Lấy kết quả nguyên trong phép chia dùng phép toán //.
- Lấy phần dư trong phép chia dùng phép toán %.
- Sử dụng phép toán ** để tính lũy thừa.

- Python sử dụng ký hiệu = để gán giá trị.
- Có thể gán nhiều giá trị vào nhiều biến cùng lúc.



```
Python 3.6 (32-bit)
Python 3.6.5 (v3.6.5:f59c0932b4,
Type "help", "copyright", "credit
>>> s = "Hello, World!!!"
>>> print(s)
Hello, World!!!
>>> a, b, c = 5, 9, -4
>>> print(a)
5
>>> print(b)
9
>>> print(c)
-4
```

- Một số phép gán rút gọn: +=, -=, *=, /=, %=, **=, //=.
- Ở đây, giải thích ý nghĩa +=, các phép gán còn lại tương tự. Phép gán $a += b \Leftrightarrow a = a + b$, nghĩa là lấy giá trị $a + b$, rồi gán giá trị kết quả vào a .
- Ngoài ra, ta có thể sử dụng các phép toán ++, -- để tăng, giảm giá trị biến 1 đơn vị.



Các phép toán số học

- Ngoài ra, ta có thể sử dụng ký hiệu `_` lấy kết quả phép tính trước.
- Ví dụ

```
Python 3.6 (32-bit)
Python 3.6.5 (v3.6.5:f59c0932b4, M
Type "help", "copyright", "credits
>>> 5 + 15
20
>>> 5 * 3 + (8 - 7) * 2
17
>>> 25 / 4
6.25
>>> 25 // 4
6
>>> 2 ** 5
32
>>> 18 * _
576
>>> _ - 76
500
```



Phép toán quan hệ

| Phép toán | Ý nghĩa | Ví dụ |
|--------------------|---------------------------|-------------------------------|
| <code>==</code> | So sánh bằng | <code>5 == 5</code> → true |
| <code>!=</code> | So sánh khác nhau | <code>5 != 5</code> → false |
| <code>></code> | So sánh lớn hơn | <code>5 > 3</code> → true |
| <code>>=</code> | So sánh lớn hơn hoặc bằng | <code>5 >= 5</code> → true |
| <code><</code> | So sánh nhỏ hơn | <code>5 < 3</code> → false |
| <code><=</code> | So sánh nhỏ hơn hoặc bằng | <code>5 <= 5</code> → true |

Phép toán luận lý

| Phép toán | Ý nghĩa |
|-----------|--|
| and | a and b là đúng khi và chỉ khi a và b cùng đúng. |
| or | a or b chỉ sai khi và chỉ khi a và b đều sai. |
| not | not a là đúng nếu a sai và ngược lại. |

Chuỗi

- Chuỗi có thể đặt giữ cặp dấu **nháy đơn** ('chuỗi'), **nháy kép** ("chuỗi") hoặc **nháy ba** (""""chuỗi""" hoặc '''chuỗi''').

```
Python 3.6 (32-bit)
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46)
Type "help", "copyright", "credits" or "license" for more
>>> 'python'
'python'
>>> "python's extensive"
"python's extensive"
>>> 'python\'s extensive'
"python's extensive"
>>> """Python is high-level programming language
... "great"
... """
'Python is high-level programming language\n"great"\n'
>>> print('Python's extensive
... Python's interpreter
... ')
Python's extensive
Python's interpreter
```

- Dùng phép toán **+** để cộng hai chuỗi.
- Dùng phép toán ***** để lặp lại một chuỗi.

```
Python 3.6 (32-bit)
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 1
Type "help", "copyright", "credits" or "license
>>> s = "Hello"
>>> s = s + " World" "!!!"
>>> print(s)
Hello World!!!
>>> s*3
'Hello World!!!Hello World!!!Hello World!!!'
```

```
Python 3.6 (32-bit)
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 1
Type "help", "copyright", "credits" or "license
>>> n = 5
>>> m = 10
>>> print(("*" * m + "\n") * n)
*****
*****
*****
*****
*****
```

- Dùng số chỉ mục để lấy ký tự trong chuỗi, nếu **chỉ mục âm** thì tính từ bên phải chuỗi.
- Lấy một chuỗi con từ vị trí i đến j trong chuỗi str: **str[i:j]**
- Lấy một chuỗi con từ vị trí i đến cuối chuỗi: **str[i:]**
- Lấy chuỗi con đầu đến vị trí thứ i trong chuỗi: **str[:i]**

▪ Ví dụ

```
Python 3.6 (32-bit)
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 1
Type "help", "copyright", "credits" or "license
>>> str = "Python is a powerful language"
>>> str[2]
't'
>>> str[-5]
'g'
>>> str[12:21]
'powerful '
>>> str[10:]
'a powerful language'
>>> str[12:-9]
'powerful'
>>> str[:6]
'Python'
>>> str[:6] + "'s " + str[10:]
"Python's a powerful language"
```

▪ Sử dụng phương thức str.format() như sau:

```
>>> a = 15
>>> b = 9
>>> str.format("{} + {} = {}", a, b, a + b)
'15 + 9 = 24'
```

▪ Sử dụng chỉ số giữa {} để chỉ định vị trí đối tượng truyền vào hoặc sử dụng tên của đối số hoặc kết hợp cả hai.

```
>>> a = 15
>>> b = 9
>>> str.format("{2} = {0} - {1}", a, b, a - b)
'6 = 15 - 9'
>>> str.format("{re} = {a} - {b}", a=a, b=b, re=a-b)
'6 = 15 - 9'
>>> str.format("{re} = {0} - {1}", a, b, re=a-b)
'6 = 15 - 9'
```



Định dạng chuỗi

- Ngoài ra, ta có thể kết hợp dấu : để chỉ định một định dạng cho hiển thị giá trị của đối số, chẳng hạn giới hạn số chữ số thập phân của số thực.

```
>>> a = 11
>>> b = 3
>>> str.format("{0} / {1} = {re:.2f}", a, b, re=a/b)
'11 / 3 = 3.67'
```

- Ta cũng có thể truyền vào một kiểu dữ liệu từ điển và dùng [] để lấy giá trị của một key ra.

```
>>> price = {"orange": 20, "apple": 50, "banana": 18}
>>> str.format("Orange = {0[orange]}; Banana = {0[banana]}", price)
'Orange = 20; Banana = 18'
```



Các phương thức khác

- s.capitalize(): chuyển ký tự đầu chuỗi thành hoa.
- s.center(width[, fillchar]): trả về chuỗi chiều dài width (nếu width > len(s)), canh chuỗi s ban đầu vào giữa và hai bên điền các ký tự fillchar.
- s.count(sub[, start, end]): đếm số lần sub xuất hiện trong s tính từ start đến end.

```
s = "python is simple"
print(s.capitalize()) #Python is simple
print(s.center(30, "-")) #-----python is simple-----
print(s.count("i", 5, 12)) #2
```



Các phương thức khác

- `s.find(sub[, start, end])`: trả về vị trí tìm thấy sub trong s tính từ start đến end, và trả về -1 nếu không tìm thấy.
- `s.strip([chars])`: xoá chars ở đầu và cuối chuỗi s, mặc định là xoá khoảng trắng.
- `s.replace(old, new[, max])`: thay chuỗi old bằng new, max để chỉ số lần thay thế (mặc định là 1).
- `s.split(separator[, num])`: tạo danh sách các chuỗi con tách từ s, ký hiệu cắt là separator (mặc định khoảng trắng), num quy định số lần cắt.



Các phương thức khác

```
s = "python is simple, python is great"
print(s.find("is")) #8
print(s.find("is", 1, 7)) #-1

print(s.strip("*"))
#python is simple, python is great

print(s.replace("python", "Py"))
#*Py is simple, Py is great*
print(s.replace("python", "Py", 1))
#*Py is simple, python is great*

print(s.split(" "))
#['python', 'is', 'simple,', 'python', 'is', 'great*']
print(s.split(" ", 2))
#['python', 'is', 'simple, python is great*']
```

- if/elif/else

```
x = int(input("Nhập x = "))
if (x > 0):
    print('%s là số dương' % x)
elif (x < 0):
    print('%s là số âm' % x)
else:
    print('là số 0');
```

- Các lệnh có thể lồng các khối lệnh if/elif/else vào nhau.
- Ví dụ: chương trình giải và biện luận phương trình bậc nhất.

```
a = int(input("Nhập a = "))
b = int(input("Nhập b = "))
if a==0:
    if b==0:
        print("Phương trình vô số nghiệm")
    else:
        print("Phương trình vô nghiệm")
else:
    print("Phương trình có nghiệm duy nhất x = ", -b/a)
```


▪ for

```
# In các phần tử trong danh sách animals
animals = ["dog", "chicken", "tiger"]
for animal in animals:
    print(animal)
# In 10 số từ 0 -> 9
for i in range(10):
    print(i)
# In 5 số từ 5 -> 9
for i in range(5, 10):
    print(i)
# In các số chẵn 2 -> 10
for i in range(2, 11, 2):
    print(i)
```

▪ while

```
# Đếm số chữ số của số nguyên n
n = int(input("Nhập n = "))
if (n < 10): n = -n

dem = 0
while (n):
    dem = dem + 1
    n = n//10

print("Số chu số là", dem)
```

- for/else: khối lệnh else được thực hiện khi chạy hết danh sách cần duyệt, và sẽ không thực hiện nếu for được kết thúc bằng break.
- while/else: khối lệnh else được thực hiện khi điều kiện trong while là sai.

▪ Ví dụ

```
# Tìm các số nguyên tố 2 -> 19
for n in range(2, 20):
    for x in range(2, n):
        if n % x == 0:
            print("%d không là số nguyên tố" % n)
            break;
    else:
        print("%d là số nguyên tố" % n)

# Nhập số âm để kết thúc chương trình
while int(input("Nhập một số không âm")) >= 0:
    print("Số nhập hợp lệ")
else:
    print("Số bạn nhập vào không hợp lệ")
```



Chú ý trong biểu thức điều kiện

- Toán tử **in** và **not in** để kiểm tra một giá trị có thuộc một dãy hay không.

```
s = input("Enter your fruit: ")
if s in ["apple", "banana", "orange"]:
    print("Great")

if s not in ["apple", "banana", "orange"]:
    print("Sorry")
```

- Toán tử **is** và **is not** để kiểm tra hai đối tượng có giống nhau không.



Chú ý trong biểu thức điều kiện

- Trong Python, ta có thể thực hiện một chuỗi so sánh, chẳng hạn: $a \leq b < c == d$
- Phép so sánh với toán tử bool (**not**, **and**, **or**) thì **not** có độ ưu tiên cao nhất, **or** có độ ưu tiên thấp nhất, chẳng hạn:

$a \text{ and not } b \text{ or } c$

sẽ tương đương

$(a \text{ and } (\text{not } b)) \text{ or } c$



Chú ý trong biểu thức điều kiện

- Các kiểu dãy có thể so sánh với nhau cùng kiểu. Phép so sánh được thực hiện theo thứ tự từ điển: so sánh các cặp phần tử từ trái sang phải nếu khác nhau thì quyết định kết quả so sánh.

Ví dụ: $(1, 2) < (1, 3) \rightarrow \text{True}$

$[2, 3] < [2, 1] \rightarrow \text{False}$

$(1, 2) == (1.0, 2.0) \rightarrow \text{True}$

$(1, "a", 2.0) <= (1, "b") \rightarrow \text{True}$



Thư viện math

- `math.sqrt(x)`: tính căn bậc hai của x.
- `math.log(x)`: tính $\log(x)$
- `math.pow(x, y)`: tính x^y .
- `math.exp(x)`: tính e^x .
- `math.modf(x)`: tách phần nguyên và phần lẻ của x.
- `math.ceil(x)`: trả về số nguyên nhỏ nhất lớn hơn x.
- `math.floor(x)`: trả về số nguyên lớn nhất nhỏ hơn x.



Thư viện math

- `abs(x)`: tính trị tuyệt đối của `x`.
- `max(x1, x2, ...)`: giá trị lớn nhất trong các số.
- `min(x1, x2, ...)`: giá trị nhỏ nhất trong các số.
- `round(x[, n])`: làm tròn `x` lấy `n` số lẻ.



Thư viện random

- Thư viện `random` cung cấp các phương thức sinh số ngẫu nhiên.
- Các phương thức thư viện `random`
 - `randrange([start,]stop[, step])`: sinh ngẫu nhiên số nguyên trong `(start, stop)`.
 - `random()`: sinh ngẫu nhiên số thực trong `(0, 1)`.
 - `uniform(x, y)`: sinh ngẫu nhiên số thực trong `(x, y)`

```
import time

# Lấy thời gian hiện tại
print(time.time()) #1562469282.721617

# Lấy các thành phần của thời điểm hiện tại
t = time.localtime(time.time())
print(t) #time.struct_time(tm_year=2019, tm_mon=7, tm_mday=7,
tm_hour=10, tm_min=14, tm_sec=42, tm_wday=6, tm_yday=188,
tm_isdst=0)
print(t.tm_hour) #10
print(t.tm_min) #14

# Lấy chuỗi định dạng thời gian
print(time.asctime(t)) #Sun Jul 7 10:14:42 2019
```

```
from datetime import datetime

# Lấy thời điểm hiện tại
print(datetime.now()) #2019-07-07 10:29:53.116593

# Tạo ngày tháng
d = datetime(2019, 9, 25) #2019-09-25 00:00:00
print(d)

# Xuất chuỗi ngày tháng theo định dạng
print(d.strftime("%m/%d/%Y")) #09/25/2019
print(d.strftime("%d-%m-%y")) #25-09-19
```

```
import calendar

# Kiểm tra năm nhuận
print(calendar.isleap(2020)) # True

# Lấy danh sách ngày (theo tuần) của tháng/năm
# [[1, 2, 3, 4, 5, 6, 7], [8, 9, 10, 11, 12, 13, 14], [15, 16, 17, 18, 19, 20, 21], [22, 23, 24, 25, 26, 27, 28], [29, 30, 31, 0, 0, 0, 0]] → 0 là ngày không thuộc tháng
print(calendar.monthcalendar(2019, 7))

# Lấy thứ của ngày → 0:Monday, 1:Tuesday, ...
print(calendar.weekday(2019, 7, 7)) #6

# Lấy số ngày của tháng
print(calendar.monthrange(2019, 7))
# (0, 31) -> (thứ, số ngày)
```

- Dùng từ khóa def để định nghĩa hàm

```
def <tên_hàm>([các_tham_số]):  
    <thân-hàm>
```

- Ví dụ hàm in các số Fibonacci nhỏ hơn n

```
def fibonacci(n):  
    a, b = 0, 1  
  
    while b < n:  
        print(b)  
        a, b = b, a + b
```

- Ví dụ hàm giải và biện luận phương thức bậc nhất

```
def giai_pt_bac_1(a, b):  
    if a == 0:  
        if b == 0:  
            print("Phương trình vô số nghiệm")  
        else:  
            print("Phương trình vô nghiệm")  
    else:  
        print("Phương trình có nghiệm x = ", -b/a)
```

- Gọi sử dụng hàm theo một trong các cách sau

```
giai_pt_bac_1(2, 3)  
giai_pt_bac_1(a=2, b=3)  
giai_pt_bac_1(b=3, a=2)
```

- Tham số với giá trị mặc định



- Tạo hàm với số lượng tham số bất kỳ với ký pháp * trước tên tham số.
- Chỉ có một tham số loại này trong một hàm.
- Tham số này phải đứng sau cùng trong danh sách tham số của hàm.

- Ví dụ chương trình tính tổng dãy số

```
def tinh_tong_day(*args):  
    tong = 0  
    for a in args:  
        tong += a  
  
    return tong
```

- Gọi hàm

```
print(tinh_tong_day(1, 2))  
print(tinh_tong_day(1, 2, 9))  
print(tinh_tong_day(1, 2, 5, -3, 5))
```



Hàm lambda

- Hàm lambda là hàm ẩn danh không có tên hàm, chỉ có danh sách tham số và phần thực thi.
- Cú pháp:

```
lambda [tham-số]: <chương-trình>
```

- Ví dụ

```
k = lambda x, y: x ** y  
print(k(2, 5))  
print(k(3, -2))
```



Một số hàm có sẵn

- map(): tạo một sequence mới theo một phương thức nào đó.

```
a = [1, 2, 3]  
c = map(lambda x: x**2, a)  
print(c) #[1, 4, 9]  
  
b = [4, 5, 6]  
c = map(lambda x, y: x + y, a, b)  
print(c) #[5, 7, 9]
```



Một số hàm có sẵn

- `filter()`: lọc các phần tử trong một sequence thoả một điều kiện nào đó.
- `reduce()`: tính kết quả của các phần tử trong một sequence theo một phương thức nào đó.

```
a = [4, 5, 2]
c = filter(lambda x: x%2==0, a)
print(c) #[4, 2]

c = reduce(lambda x, y:x+y, a)
print(c) #11
```



Danh sách (List)

- Một danh sách gồm nhiều phần tử cách nhau bằng dấu phẩy.

Ví dụ: `a = [1, 2, 3]`

- Các phần tử trong danh sách có thể có kiểu dữ liệu khác nhau.

Ví dụ: `b = [7, 8, "a", "b", 5.0, 6.0]`

- Để lấy một phần tử trong danh sách hoặc cắt một đoạn trong danh sách, thực hiện tương tự như trên chuỗi. Ví dụ: `a[1] -> 2`, `b[2] -> "a"`



Danh sách (List)

- Có thể gán giá trị mới cho một phần tử trong danh sách (**mutable**). Ví dụ: `a[2] = 5`
- Dùng phép toán `+` để nối hai danh sách.
- Dùng hàm `len` lấy chiều dài danh sách.
- Dùng hàm `del` xóa một phần tử trong danh sách.

```
Python 3.6.5 (v3.6.5:f59c093)
Type "help", "copyright", "c
>>> a = [5, 8]
>>> b = ["Chin", 8, "Muoi"]
>>> c = a + b
>>> c
[5, 8, 'Chin', 8, 'Muoi']
>>> len(c)
5
>>> del c[2]
>>> c
[5, 8, 8, 'Muoi']
```



Các phương thức của danh sách

| Phương thức | Giải thích |
|--------------------------------|---|
| <code>list.append(x)</code> | Thêm một phần tử vào cuối danh sách, tương đương <code>list[len(a):] = [x]</code> |
| <code>list.extend(L)</code> | Thêm các phần tử vào danh sách từ một danh sách khác. Tương đương <code>list[len(a):] = L</code> |
| <code>list.insert(i, x)</code> | Chèn phần tử <code>x</code> vào vị trí <code>i</code> |
| <code>list.remove(x)</code> | Xóa phần tử <code>x</code> đầu tiên trong danh sách |
| <code>list.pop([i])</code> | Trả về phần tử tại vị trí được chỉ định và đồng thời xóa nó khỏi danh sách. Nếu không chỉ định <code>i</code> , thì phần tử cuối danh sách sẽ được chọn |



Các phương thức của danh sách

| Phương thức | Giải thích |
|---|--|
| <code>list.clear()</code> | Xóa tất cả phần tử trong danh sách, tương đương <code>del list[:]</code> |
| <code>list.index(x)</code> | Trả về vị trí đầu tiên x trong danh sách |
| <code>list.count(x)</code> | Trả về số lần x xuất hiện trong danh sách |
| <code>list.sort(key=None, reverse=False)</code> | Sắp xếp các phần tử trong danh sách |
| <code>list.reverse()</code> | Đảo ngược danh sách |
| <code>list.copy()</code> | Sao chép danh sách, tương đương <code>list[:]</code> |



Các phương thức của danh sách

▪ Ví dụ

```
>>> list = [1, 2]
>>> list.append(3)
>>> list
[1, 2, 3]
>>> list2 = [2, 5]
>>> list.extend(list2)
>>> list
[1, 2, 3, 2, 5]
>>> list.remove(2)
>>> list
[1, 3, 2, 5]
>>> list.pop()
5
>>> list.pop(2)
2
>>> list
[1, 3]

>>> list = [1, 2, 3, 2, 2]
>>> list.index(2)
1
>>> list.count(2)
3
>>> list.sort()
>>> list
[1, 2, 2, 2, 3]
>>> list.reverse()
>>> list
[3, 2, 2, 2, 1]
>>> c = list.copy()
>>> c
[3, 2, 2, 2, 1]
>>> c.clear()
>>> c
[]
>>> list
[3, 2, 2, 2, 1]
```



Sử dụng danh sách như stack

- Có thể sử dụng hai phương thức append và pop trên danh sách để thao tác giống như một stack.
- `append(x)` → chèn x vào cuối danh sách.
- `pop()` → lấy phần tử cuối danh sách và xóa nó khỏi danh sách.



Sử dụng danh sách như stack

▪ Ví dụ

```
>>> stack = []
>>> stack.append(5)
>>> stack.append(8)
>>> stack.append(2)
>>> stack.append(9)
>>> stack
[5, 8, 2, 9]
>>> stack.pop()
9
>>> stack.pop()
2
>>> stack.pop()
8
>>> stack
[5]
```



Sử dụng danh sách như queue

- Dùng phương thức `append` và `collections.deque` để thao tác danh sách giống như queue

```
>>> from collections import deque
>>> queue = deque([])
>>> queue.append(10)
>>> queue.append(15)
>>> queue.append(27)
>>> queue.append(45)
>>> queue.append(52)
>>> queue.popleft()
10
>>> queue.popleft()
15
>>> queue.popleft()
27
>>> queue
deque([45, 52])
```



Bộ (Tuple)

- Tuple chứa các giá trị cách nhau bởi dấu phẩy (,). Tuple có thể bao gồm dấu ngoặc "(" , ")" hoặc không. Ví dụ:

```
tuple1 = 1, 2.0, "hello"
```

```
tuple2 = ("apple", "lemon")
```

- ***Không thể thay đổi giá trị*** một phần tử tuple (***immutable***). Tuy nhiên có thể tạo tuple với các đối tượng thay đổi.



Bộ (Tuple)

- Nếu tuple khai báo không có phần tử nào thì phải dùng dấu ().

Ví dụ: `empty = ()`

- Nếu tuple khai báo chỉ có một phần tử thì phải có dấu phẩy ở cuối.

Ví dụ: `singleton= ("apple",)`

hoặc `singleton = "lemon",`



Tập hợp (Set)

- Set là một tập **không có thứ tự** và **không có phần tử trùng nhau**.
- Sử dụng `{}` hoặc hàm **`set()`** để tạo một tập hợp.
- Khi tạo tập hợp rỗng phải sử dụng `set()`.

Tập hợp (Set)

- Dùng **-** để trừ hai tập hợp.

Ví dụ: $\{1, 2\} - \{2, 3\} = \{1\}$

- Dùng **|** để lấy hợp hai tập hợp.

Ví dụ: $\{1, 2\} | \{2, 3\} = \{1, 2, 3\}$

- Dùng **&** để lấy giao hai tập hợp.

Ví dụ: $\{1, 2\} \& \{2, 3\} = \{2\}$

- Dùng **^** để lấy phần tử chỉ thuộc một trong hai tập hợp.

Ví dụ: $\{1, 2\} \wedge \{2, 3\} = \{1, 3\}$

Tập hợp (Set)

- Ví dụ

```
>>> animals = {"cat", "dog", "chicken", "tiger",  
               "cat", "chicken"}  
>>> print(animals)  
{'cat', 'tiger', 'chicken', 'dog'}  
>>> a = set("aabbcccaadde")  
>>> b = set("cmmnnnee")  
>>> a  
{'c', 'a', 'b', 'd', 'e'}  
>>> b  
{'c', 'm', 'n', 'e'}  
>>> a - b  
{'a', 'b', 'd'}  
>>> a | b  
{'d', 'c', 'e', 'n', 'a', 'b', 'm'}  
>>> a & b  
{'c', 'e'}  
>>> a ^ b  
{'d', 'n', 'm', 'b', 'a'}
```



Từ điển (Dictionary)

- **Dictionary** là một tập hợp các **cặp key/value không có thứ tự**, key bắt buộc phải là duy nhất. Các cặp key/value cách nhau bằng dấu phẩy (,)
- Ví dụ:

```
user = {  
    "first_name": "Thanh",  
    "last_name": "Duong"  
}
```



Từ điển (Dictionary)

- Truy xuất hoặc gán giá trị phần tử trong từ điển thông qua key. Ví dụ:

```
user["gender"] = "Male";  
user["first_name"] -> "Thanh"
```

- Để lấy danh sách key trong từ điển, dùng phương thức `keys()`. Ví dụ: `user.keys()`

→ ["first_name", "last_name", "gender"]



Từ điển (Dictionary)

- Để xóa một cặp key/value trong từ điển, dùng del

Ví dụ: `del user["last_name"]`

- Để kiểm tra một key có trong từ điển không sử dụng toán tử **in**.

Ví dụ: `"last_name" in user -> True`



Từ điển (Dictionary)

```
>>> user = {"first_name": "Thanh", "last_name": "Duong"}
>>> user["gender"] = "Male"
>>> user
{'last_name': 'Duong', 'first_name': 'Thanh', 'gender': 'Male'}
>>> user["first_name"]
'Thanh'
>>> user.keys()
dict_keys(['last_name', 'first_name', 'gender'])
>>> del user["gender"]
>>> user
{'last_name': 'Duong', 'first_name': 'Thanh'}
>>> "first_name" in user
True
>>> "gender" in user
False
>>> courses = dict(math=10, phy=9, geo=8)
>>> courses
{'math': 10, 'geo': 8, 'phy': 9}
>>> cell_phone = dict([("price", 10), ("color", "blue")])
>>> cell_phone
{'price': 10, 'color': 'blue'}
```



Một số kỹ thuật lặp

```
# Lặp lấy key/value trên dictionary
user = {
    "first_name": "Thanh",
    "last_name": "Duong",
    "gender": "Male"
}
for key, value in user.items():
    print(key, ":", value)
```

```
# Lặp lấy chỉ số danh sách
animals = ["bird", "dog", "cat"]
for idx, value in enumerate(animals):
    print("Animal", idx + 1, ":", value)
```



Một số kỹ thuật lặp

```
# Lặp song song trên các danh sách
animals_vn = ["Con chim", "Con chó", "Con mèo"]
for en, vi in zip(animals, animals_vn):
    print("In English:", en, "; In Vietnamese:", vi)
```

```
animals = ["bird", "dog", "cat"]
```

```
# Đảo ngược danh sách
for animal in reversed(animals):
    print(animal)
```

```
# Sắp xếp danh sách theo alphabetical
for animal in sorted(animals):
    print(animal)
```




Module

- Module là một **tập tin python** (đuôi **`*.py`**) chứa các câu lệnh và định nghĩa python.
- Module cho phép **tái sử dụng** mã nguồn, chương trình được chia nhỏ nên **dễ bảo trì** hơn.
- Tên của một module được gán trong biến toàn cục tên là `__name__`
- Ta có thể **import** một module vào module khác sử dụng bằng cách dùng **import** hoặc kết hợp **from/import**.



Module

- Ví dụ: ta có gói (python package) tên là `example`, và có hai module trong gói này là `demo.py` và `func.py`
 - Trong `func.py` có 2 hàm: đếm số chữ số và tính tổng số chữ số của số nguyên n.
 - Trong `demo.py` ta sẽ import `func` và gọi các hàm thực thi

```
+ example
- func.py
- demo.py
```

- Ví dụ ta có module func.py như sau:

```
# func.py
def dem_so_chu_so(n):
    dem = 0
    while (n):
        dem = dem + 1
        n = n//10

    return dem

def tong_chu_so(n):
    tong = 0
    while (n):
        tong = tong + n%10
        n = n//10

    return tong
```

- Gọi sử dụng các chức năng trong func.py

```
# demo.py
# import func module từ gói example (python package)
from example import func

print(func.__name__)
n = int(input("Nhập n = "))

# Gọi hàm thực thi
print("Số chu so của %d:" % n, func.dem_so_chu_so(n))
print("Tong so chu so của %d:" % n, func.tong_chu_so(n))

# Có thể gán một hàm vào một biến
f1 = func.dem_so_chu_so
f2 = func.tong_chu_so
print("Số chu so của %d:" % n, f1(n))
print("Tong so chu so của %d:" % n, f2(n))
```

- Nếu trong module có các câu lệnh (nhằm mục đích khởi động cho module) thì nó chỉ thực thi lần đầu tiên khi module được import.
- Ta có thể import trực tiếp các hàm trong module.

```
from example.func import dem_so_chu_so, tong_chu_so
```

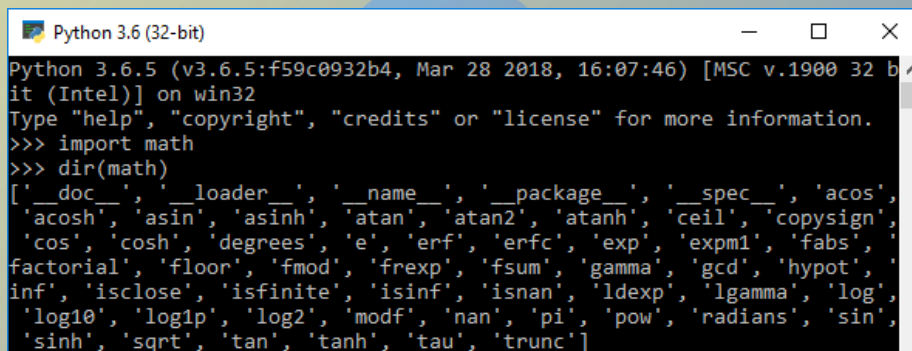
- Để import tất cả các hàm trong module ta dùng import *.

```
from example.func import *
```

- Chú ý với lệnh import này những hàm có tên bắt đầu bằng "_" sẽ không được import.
- Chẳng hạn module func.py có hàm `_in_ket_qua` thì với lệnh `import *` ở trên, hàm này sẽ không được import vào.

- Khi import một module thì thứ tự python tìm module này như sau:
 - Tìm trong các module đã xây dựng sẵn (built-in modules)
 - Tìm trong các thư mục được cho bởi `sys.path`. `sys.path` được khởi động từ các vị trí sau:
 - Thư mục chứa script thực thi hoặc thư mục hiện tại nếu không có tập tin chỉ định
 - Trong biến môi trường PYTHONPATH
 - Những thư viện mặc định có sẵn nếu PYTHONPATH chưa có giá trị

- Sử dụng hàm có sẵn `dir()` để xem danh sách các biến, tên hàm xây dựng trong một module.



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 b
it (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', '
factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', '
inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```




Thực thi Module

- Ta có thể thực thi module python trên command prompt bằng lệnh như sau:

```
python <tên_module>.py [<các-đối-số>]
```

- Thêm các dòng code sau vào cuối module func.py

```
if __name__ == "__main__":  
    import sys  
    n = int(sys.argv[1])  
    print("So chu so:", dem_so_chu_so(n))  
    print("Tong cac chu so:", tong_chu_so(n))
```

```
C:\Users\dhthanh>python PycharmProjects\example\example\func.py 12345  
So chu so: 5  
Tong cac chu so: 15
```



Package

- Package được xem là module namespace, bao gồm có nhiều module.
- Ta có thể sử dụng hàm `dir` (được xây dựng sẵn trong python) để xem tên các module trong package.
- Mỗi package bắt buộc phải có tập tin `__init__.py` để python biết đó là một thư mục python package. Tập tin này thường là rỗng, nhưng ta có thể sử dụng nó để khởi động các giá trị cho package.



Package

- Khi ta import một package thì python sẽ tìm kiếm các thư mục con của package thông qua các thư mục trong `sys.path`
- Khi sử dụng `from package import item` thì item có thể là
 - Một package con trong package
 - Một module trong package
 - Một hàm, lớp, biến được định nghĩa trong các module của package



Package

- Nếu trong `__init__.py` của package có định nghĩa danh sách tên các module trong package vào biến `__all__` thì các module này sẽ được import từ lệnh `from package import *`
- Nếu `__all__` không được định nghĩa thì lệnh `from package import *` sẽ không import các module trong package mà chỉ đảm bảo package được import và tất cả các tên được định nghĩa trong package.

- Lớp (class) cung cấp khả năng lập trình hướng đối tượng trong Python.
- global & nonlocal



- Cú pháp khai báo lớp trong Python

```
class ClassName:  
    # something to do
```

- Cú pháp khai báo phương thức khởi tạo của lớp

```
class ClassName:  
    def __init__(self, *args, **kwargs):  
        # to do something
```

- Tạo thể hiện của lớp dùng tên lớp và truyền vào danh sách đối số của phương thức khởi tạo `__init__`

```
x = ClassName([các_đối_số])
```

- Ta truy cập vào các thuộc tính và phương thức của lớp sử dụng toán tử `"."`

```
x.<tên_thuộc_tính>  
x.<tên_phương_thức>([các_đối_số])
```

- Ví dụ

```
p = PhanSo(5, 4)  
p.hien_thi()  
q = PhanSo(2, 3)  
q.hien_thi()
```

```
class PhanSo:  
    tu_so = 0  
    mau_so = 1  
  
    def __init__(self, tu, mau):  
        self.tu_so = tu  
        self.mau_so = mau  
  
    def hien_thi(self):  
        print(str.format("{} / {}",  
                           self.tu_so, self.mau_so))
```


- Python quy ước khai báo thành viên bắt đầu bằng
 - 2 dấu gạch chân liên tiếp là thành viên private.
 - 1 dấu gạch chân là thành viên non-public.



- Khai báo thành viên tĩnh

```
class A:  
    @staticmethod  
    def method1():  
        pass  
  
    @staticmethod  
    def method2():  
        pass
```

- Gọi sử dụng thành viên tĩnh

```
A.method1()  
A.method2()
```

- Cú pháp khai báo kế thừa

```
class ClassName(BaseClassName):  
    # something to do
```

- Đa kế thừa

```
class ClassName(BaseClass1, BaseClass2):  
    # something to do
```

```
class NhanVien:  
    def __init__(self, ho_ten, gioi_tinh, que_quan):  
        self.ho_ten = ho_ten  
        self.gioi_tinh = gioi_tinh  
        self.que_quan = que_quan  
  
    def hien_thi(self):  
        print("Họ tên: ", self.ho_ten)  
        print("Giới tính: ", self.gioi_tinh)  
        print("Quê quán: ", self.que_quan)  
  
class GiangVien(NhanVien):  
    def __init__(self, ho_ten, gioi_tinh, que_quan, hoc_vi):  
        super().__init__(ho_ten, gioi_tinh, que_quan)  
        self.hoc_vi = hoc_vi  
  
    def hien_thi(self):  
        super().hien_thi()  
        print("Học vị: ", self.hoc_vi)
```



Lớp trừu tượng

- Cú pháp khai báo lớp trừu tượng

```
from abc import ABC, abstractmethod

class A(ABC):
    @abstractmethod
    def method1(self):
        pass
```



Xử lý ngoại lệ

- Các lỗi xảy ra trong quá trình thực thi chương trình gọi là ngoại lệ (exception).
- Xử lý ngoại lệ tổng quát:

```
try:
    # code khối try
except (ex1, ex2, ...):
    # code khối except
else:
    # code khối else
finally:
    # code khối finally
```



Xử lý ngoại lệ

- Nếu trong khối **try** không có ngoại lệ nào xảy ra thì các khối except sẽ bị bỏ qua.
- Nếu có ngoại lệ xảy ra trong khối try, thì phần code còn lại trong try sẽ bị bỏ qua và tiếp các ngoại lệ được bắt trong các khối except khớp với ngoại lệ xảy ra và thực thi trong khối except đó. Trường hợp nếu không có except nào phù hợp thì chương trình sẽ dừng lại.



Xử lý ngoại lệ

- Khối lệnh **else** sau try cũng sẽ được thực thi nếu trong try không có ngoại lệ nào được sinh ra.
- Khối lệnh **finally** luôn được thực thi, bất chấp có ngoại lệ xảy ra trong khối try hay không.

- Sử dụng câu lệnh **raise** để sinh một ngoại lệ.
- Đối tượng cần truyền cho raise là thể hiện của Exception hoặc các lớp con của Exception. Nếu ta truyền vào lớp ngoại lệ cho raise thì nó ngầm định sẽ gọi phương thức khởi tạo không tham số của lớp ngoại lệ đó.

```
raise Exception("Something is wrong.")
raise ValueError("The value is wrong.")
raise NameError
```

- Ghi tập tin csv

```
import csv

with open("demo.csv", "w", newline="") as f:
    writer = csv.DictWriter(f, fieldnames=["fn", "ln"])
    writer.writeheader()
    writer.writerow({"fn": "Thanh", "ln": "Duong"})
    writer.writerow({"fn": "Tran", "ln": "Nguyen"})
    writer.writerow({"fn": "Phuong", "ln": "Le"})
```

```
with open('demo2.csv', 'w', newline='') as f:
    writer = csv.writer(f, delimiter=',',
                        quotechar='|',
                        quoting=csv.QUOTE_MINIMAL)
    writer.writerow(["Thanh", "Duong"])
    writer.writerow(["Tran", "Nguyen"])
    writer.writerow(["Phuong", "Le"])
```



Đọc ghi tập tin CSV

- Đọc ghi tập tin csv

```
import csv

with open('demo.csv', newline='') as csvfile:
    reader = csv.reader(csvfile, delimiter=',', quotechar='|')
    for row in reader:
        print(row)
```

```
import csv

with open("demo.csv", newline="") as f:
    reader = csv.DictReader(f)
    for row in reader:
        print("%s %s" % (row["fn"], row["ln"]))
```



Làm việc với JSON

- JSON (JavaScript Object Notation)
- `json.dump(json_obj, file)`: serialize obj → json stream

```
import json
user = {
    "name": "Hữu Thành",
    "username": "thanhduong",
    "password": "123456"
}
f = open("user.json", "w", encoding="utf-8")
json.dump(user, f, ensure_ascii=False, indent=4)
f.close()
```

```
1 {
2     "name": "Hữu Thành",
3     "username": "thanhduong",
4     "password": "123456"
5 }
```



Làm việc với JSON

- `json.dumps()`: serialize obj → json str

```
user = {
    "name": "Hữu Thành",
    "username": "thanhduong",
    "password": "123456"
}

d = json.dumps(user, indent=4)
print(d)
```



Làm việc với JSON

- `json.load()`: deserialize stream → obj

```
import json
f = open("user.json", encoding="utf-8")
print(json.load(f))
f.close()
```

- `json.loads()`: deserialize str → obj

```
s = """
{
    "username": "thanhduong",
    "password": "123456"
}
"""

d = json.loads(s)
print(d["username"])
```