

Bài 03: Lập trình Python cơ bản (tiếp)

AI Academy Vietnam

1. Hàm trong Python

- a. Giới thiệu Hàm | Xây dựng hàm trong Python
- b. Return | Tham số của hàm | Phạm vi của biến
- c. Hàm đệ quy | Hàm ẩn danh (Lambda)

2. Lớp và đối tượng trong Python

- a. Tạo lớp – đối tượng | Truy cập phương thức, thuộc tính của đối tượng
- b. Thừa kế

3. Module và Package

- a. Giới thiệu module và cách xây dựng module
- b. Import các package sử dụng

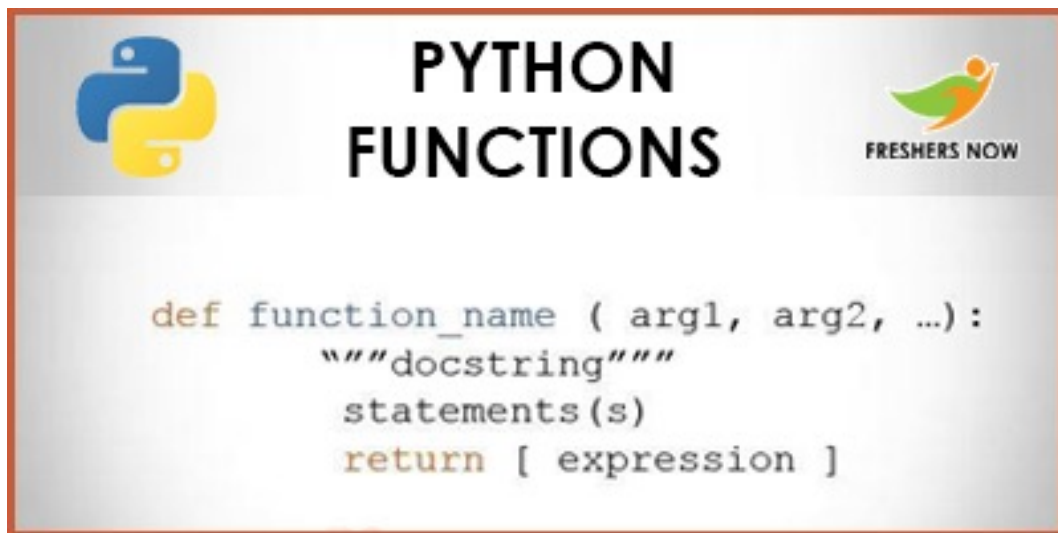
4. Đọc và ghi file với Python

5. Multithreads, Multiprocessing, Functools

1. Hàm trong Python

Cơ bản về hàm trong Python

- Hàm trong Python là một nhóm các câu lệnh trong chương trình được tổ chức chung với nhau để thực hiện một chức năng hay một nhiệm vụ cụ thể nào đó.
- Sử dụng hàm giúp phân rã chương trình từ một chương trình lớn, phức tạp thành các phần cụ thể nhỏ hơn giúp dễ quản lý, tổ chức, nâng cao khả năng tái sử dụng và chia sẻ công việc.



```
1  #Ví dụ: Xây dựng 1 Hàm thực hiện chào hỏi
2  def hello_aiv(name):
3      print('Hi ',name,', How are you?')
4      print('Have a nice day!')
```

```
1  #Sử dụng hàm đã xây dựng
2  hello_aiv('AIV')
```

Hi AIV , How are you?
Have a nice day!

1. Cú pháp:

```
def tên_hàm(các_tham_số):  
    "function_docstring"  
    Các câu lệnh xử lý bên trong hàm  
    return [kết quả trả về]
```

- Từ khóa **def** được sử dụng để bắt đầu phần định nghĩa hàm.
- sau đó là **tên_hàm**, tên hàm được đặt theo quy tắc như tên biến.
- Các tham số được truyền vào bên trong các dấu ngoặc đơn.
- Ở cuối là dấu hai chấm “:”.
- Sau đó là lệnh để được thực thi.
- Kết quả trả về cho hàm được thực hiện thông qua lệnh return

Lưu ý: Hàm không bắt buộc phải có tham số truyền vào hay kết quả trả về

- Trước khi bắt tay vào xây dựng 1 hàm trong Python, cần phải tự **trả lời các câu hỏi**:
 - Hàm này sử dụng nhằm mục đích gì?
 - Hàm này nhận đầu vào là gì?
 - Hàm này trả kết quả ra là gì?

```
1  #Xây dựng hàm tính n!  
2  #1)Hàm này dùng để làm gì? - Để tính n!  
3  #2)Hàm này nhận dữ liệu vào là gì? - Một số nguyên dương N  
4  #3)Hàm trả kết quả là gì? - Một số nguyên dương là tích của 1*2*...*N  
5  def giai_thua(n):  
6      #Nhóm câu lệnh xử lý bên trong hàm  
7      tích=1  
8      for i in range(1,n+1):  
9          tích=tích*i  
10     #kết quả trả về cho hàm  
11     return tích
```

```
1  n = int(input('Nhập vào một số nguyên N:'))  
2  print(n, '!=', giai_thua(n))
```

```
Nhập vào một số nguyên N:10  
10 != 3628800
```

Gọi hàm:

- Hàm sau khi được xây dựng, có thể thực hiện lời gọi hàm ở nơi nào cần dùng đến.

```
1 n = int(input('Nhập vào một số nguyên N:'))
2 print(n, '!=', giai_thua(n))
```

```
Nhập vào một số nguyên N:10
10 != 3628800
```

```
1 #Gọi hàm giai_thua đã xây dựng
2 #Tính 12!
3 print('12! = ', giai_thua(12))
```

```
12! = 479001600
```

- Các lệnh mà chúng ta đã được học và sử dụng trước đây như: `print()`, `input()`, `type()`, `int()`, `float()`, `str()`... Đây thực chất là các **hàm được Python định nghĩa sẵn**.

Lệnh return:

- Lệnh **return <kết quả trả về>** được sử dụng để trả kết quả xử lý thông qua tên hàm.
- Lệnh return có thể có hoặc không.
- Trong trường hợp **không cung cấp <kết quả trả về>**, thì hàm **return** này sẽ **trả về None**. Nói cách khác, lệnh return được sử dụng để thoát khỏi định nghĩa hàm.

```
1 #Hàm không có Lệnh return
2 def hello_aiv(name):
3     print('Hello ', name, ', How are you?')
4     print('Have a nice day.....!')
```

```
1 #Hàm trả về 1 giá trị
2 def giai_thua(n):
3     #nhóm các câu lệnh xử lý bên trong hàm
4     tích=1
5     for i in range(1,n+1):
6         tích=tích*i
7     #kết quả trả về cho hàm
8     return tích
```


Cơ bản về hàm trong Python

Lệnh `return()` có thể trả về 1 hay nhiều kết quả, nếu có nhiều hơn một kết quả thì ngăn cách nhau bởi **dấu phẩy**.

```
1  #Hàm trả về nhiều giá trị
2  #Hàm tính tổng, hiệu, tích, thương:
3  def all_ab(a,b):
4      add = a+b
5      sub = a-b
6      multi = a*b
7      div = a/b
8      #hàm trả về kết quả là 4 giá trị
9      return add, sub, multi, div
```

```
1  a=10
2  b=6
3  #Lấy kết quả trả về khi thực hiện hàm
4  tong,hieu,tich,thuong = all_ab(a,b)
5
6  #Lưu ý: Thứ tự trả về theo đúng thứ tự đã viết trong
7  #câu lệnh return
8  print('Tổng ',a,'+',b,'=',tong)
9  print('Hiệu ',a,'-',b,'=',hieu)
10 print('Tích ',a,'*',b,'=',tich)
11 print('Thương ',a,'/',b,'=',thuong)
```

Tổng 10 + 6 = 16

Hiệu 10 - 6 = 4

Tích 10 * 6 = 60

Thương 10 / 6 = 1.6666666666666667

Tham số truyền vào hàm:

- Tham số bắt buộc
- Tham số có mặc định (Default parameter)
- Tham số có độ dài biến (Variable-Length Parameter)

➤ Tham số bắt buộc

```
1 #Xây dựng hàm tính n!  
2 #Hàm giai_thua có 1 tham số bắt buộc n  
3 def giai_thua(n):  
4     tích=1  
5     for i in range(1,n+1):  
6         tích=tích*i  
7     #kết quả trả về cho hàm  
8     return tích
```

```
1 #Gọi hàm giai_thua đã xây dựng  
2 print('12! = ', giai_thua(12))
```

12! = 479001600

```
1 #Gọi hàm giai_thua đã xây dựng  
2 #Khi không truyền vào tham số  
3 print('12! = ', giai_thua())
```

TypeError

Traceback (most recent call last)

<ipython-input-14-01226097b9b9> in <module>

1 #Gọi hàm giai_thua đã xây dựng

2 #Khi không truyền vào tham số

----> 3 print('12! = ', giai_thua())

TypeError: giai_thua() missing 1 required positional argument: 'n'

➤ Tham số mặc định cho hàm:

- Để hạn chế trường hợp báo lỗi khi gọi hàm không cung cấp tham số thì trong Python cũng cung cấp cho chúng ta **thiết lập giá trị mặc định của tham số** khi khai báo hàm. Bằng cách sử dụng dấu **=** với cú pháp như sau:

```
def ten_ham(param = defaultValue):  
    # code
```

- Trong đó: **defaultValue** là giá trị mặc định của tham số đó mà bạn muốn gán.

➤ Ví dụ:

- Hàm `sum_ab(a,b)` gọi khi truyền tham số và và khi không truyền tham số:

```
1  #Hàm tính tổng
2  def sum_ab(a=5, b =7):
3      total = a + b
4      return total
```

```
1  #Gọi hàm sum_ab() truyền vào 2 tham số
2  print(sum_ab(8,13))
```

21

```
1  #Gọi hàm sum_ab() không truyền vào tham số
2  #Sử dụng tham số mặc định
3  print(sum_ab())
```

12

➤ Tham số có độ dài biến (Variable-Length Parameter)

- Trên thực tế, không phải lúc nào chúng ta cũng biết được chính xác số lượng biến truyền vào trong hàm. Chính vì thế trong Python có cũng cấp cho chúng ta khai báo một **param** đại diện cho các biến truyền vào hàm bằng cách thêm dấu ***** vào trước param đó.

- Ví dụ:

```
1  #Xây dựng hàm tính tổng các số đưa vào
2  def get_sum(*num):
3      tmp=0
4      #duyet các tham số
5      for i in num:
6          tmp = tmp+i
7      return tmp
8
9  #Gọi hàm và truyền các tham số cho hàm
10 result = get_sum(1,2,3,4,5)
11 print('Kết quả:', result)
```

Kết quả: 15

Phạm vi của biến.

- Một biến chỉ có tác dụng trong phạm vi mà nó khai báo (global – local).
- Khi một biến được khai báo ở trong hàm thì nó chỉ có thể được sử dụng ở trong hàm đó.

```
1 x = 300 #Biến toàn cục, có tác dụng trong toàn bộ chương trình
2 y = 800
3 def myfunc():
4     #Biến địa phương, chỉ có tác dụng trong thân hàm
5     x = 200
6     total = x + y
7     print('(Local) x :', x)
8     print('total :', total)
9 #Gọi hàm
10 myfunc()
11
12 print('-----')
13 print('(global) x:', x)
14
```

(Local) x : 200

total : 1000



(global) x: 300

```
1 #Thiết lập một biến local thành global
2 def myfunc():
3     global k #Thiết lập biến k là biến global
4     k = 300
5     print('Inside func: k = ',k)
6
7 myfunc()
8
9 print('Outside func: k =', k)
```

Inside func: k = 300

Outside func: k = 300

- Ngôn ngữ Python cho phép hàm gọi đến chính nó, người ta gọi là đệ quy hay quay lui

```
def recurse():  
    ...  
    recurse()  recursive call  
    ...  
recurse() 
```

- Trong giải thuật phải có một điều kiện dừng để đệ quy kết thúc.
- Chương trình sử dụng đệ quy thì dễ hiểu nhưng hao tốn tài nguyên CPU, ảnh hưởng đến thời gian chạy chương trình nếu số lần đệ quy của hàm lớn.

```
1 #Hàm tính n! theo phương pháp đệ quy  
2 def giai_thua(n):  
3     if n==0:          #Điều kiện dừng để kết thúc đệ quy  
4         return 1;    #vì 0! = 1 nên n==0 là vị trí kết thúc của đệ quy  
5     else:  
6         return giai_thua(n-1)*n  #1*2*....*n
```

```
1 n = int(input("Nhập vào số N:"))  
2 print(n, '!=', giai_thua(n))
```

Nhập vào số N:10
10 != 3628800

- Một hàm ẩn danh là một hàm được định nghĩa mà không có tên
- Các hàm bình thường được định nghĩa bằng từ khóa def; hàm ẩn danh được định nghĩa bằng từ khóa lambda

Cú pháp:

```
lambda arguments : Expression
```

- Các hàm lambda có thể có bất kỳ đối số nào nhưng chỉ có một biểu thức.

```
1  #Ví dụ hàm ẩn danh: 1 tham số
2  x = lambda a : a + 10
3  #lambda a: a + 10 là hàm lambda
4  #Trong đó: a là tham số truyền vào
5  #          a+10 là biểu thức
6
7  print(x(5))
8  print(x(7))
```

```
# Program to filter out only the even items from a list
my_list = [1, 5, 4, 6, 8, 11, 3, 12]
new_list = list(map(lambda x: x*2, my_list))
print(new_list)
```

```
[2, 10, 8, 12, 16, 22, 6, 24]
```

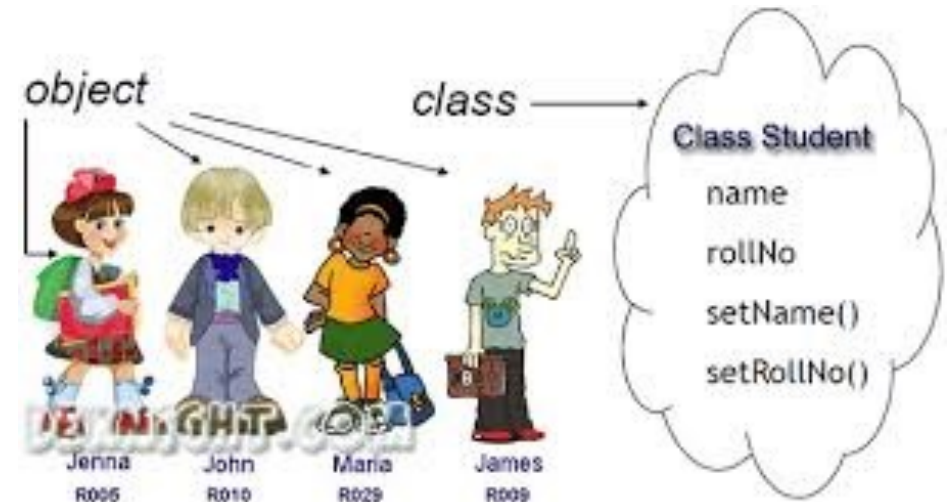
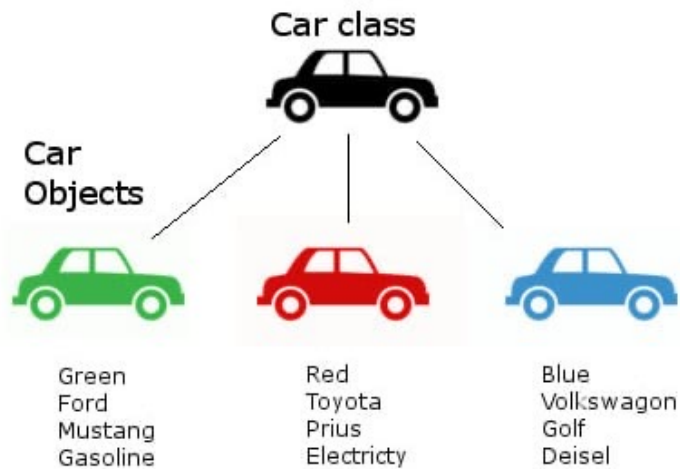



Thực hành

2. Lớp, đối tượng trong Python

1. Giới thiệu lớp

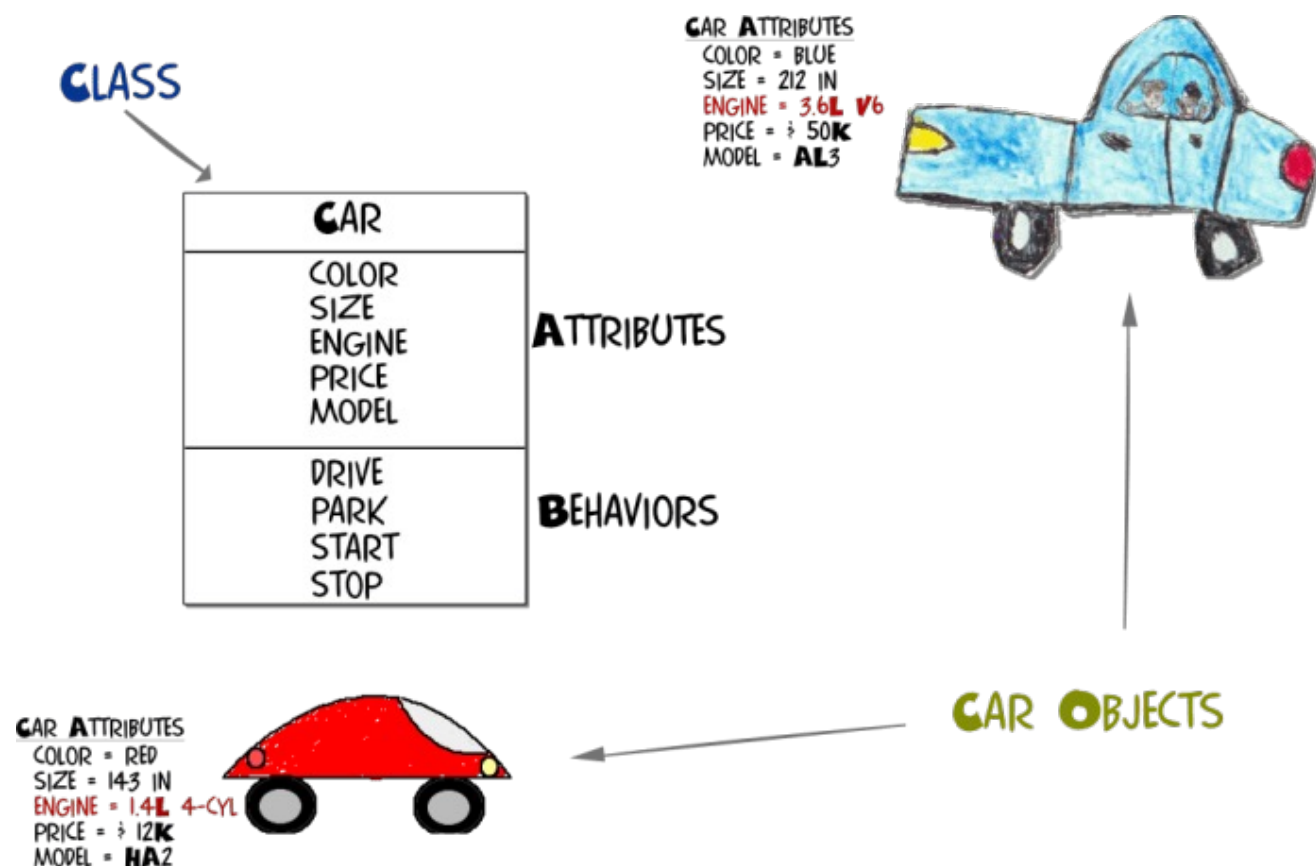
- Python là một ngôn ngữ lập trình hướng đối tượng



- Lớp (class) là một cấu trúc logic dùng để định nghĩa khuôn dạng và tính chất của các Đối tượng (Object).
- Lớp được định nghĩa như là một kiểu dữ liệu mới.

Lớp, đối tượng trong Python

- Mỗi một đối tượng có 2 thành phần chính:



- Các thuộc tính: dùng để chứa các thông tin mô tả các đặc điểm của đối tượng. Sử dụng các biến để lưu giữ những thông tin này. Khi đó các biến được gọi là các thuộc tính.
- Các phương thức: dùng để mô tả các hành vi của đối tượng. Sử dụng các hàm để mô tả các hành vi này. Khi đó các hàm được gọi là các phương thức.

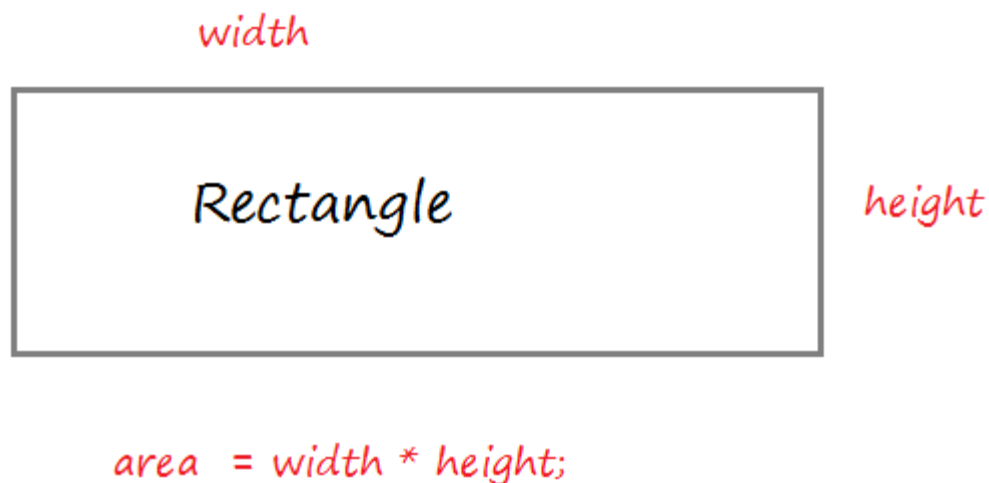
➤ Khai báo Class trong Python

```
class ClassName:  
    'Gồm các thuộc tính, phương thức'  
    # Code ...
```

- Trong đó, **className** là tên của class cần khai báo. Một số lưu ý khi đặt tên cho lớp:
 - Tên lớp nên là một danh từ,
 - Tên lớp được đặt ký tự đầu tiên của mỗi từ là in hoa.
 - Tên lớp nên đơn giản, mang tính mô tả và đầy đủ ý nghĩa
 - Tên lớp không được là một từ khóa nào đó của Python
 - Tên lớp không bắt đầu bằng số, có thể bắt đầu với dấu \$ hoặc ký tự gạch dưới.

➤ Khai báo Class: Rectangle

- có 2 thuộc tính: width, height
- 2 phương thức: getArea(), getPerimeter()



```
1  #Tạo một lớp Rectangle
2
3  class Rectangle:
4      #Lớp Rectangle có 2 thuộc tính: width, height
5
6      #Phương thức khởi tạo đối tượng (Constructor)
7      def __init__(self, width, height):
8          self.width = width
9          self.height = height
10
11     #Phương thức tính diện tích
12     def getArea(self):
13         area = round(self.width * self.height,1)
14         return area
15
16     #Phương thức tính chu vi
17     def getPerimeter(self):
18         perimeter = round((self.width + self.height)*2,1)
19         return perimeter
```

➤ Một số khái niệm hướng đối tượng

- **Thuộc tính (Attribute):** Thuộc tính là một thành viên của lớp, Hình chữ nhật có 2 thuộc tính width và height
- **Phương thức (Method):**
 - Phương thức của class tương tự như một hàm thông thường, nhưng nó là một hàm của class. Để sử dụng được cần phải gọi thông qua đối tượng.
 - Tham số đầu tiên của phương thức luôn là self (Một từ khóa ám chỉ chính class đó)
- **Phương thức khởi tạo (Constructor):**
 - Là một phương thức đặc biệt của lớp, luôn có tên là `__init__`. Tham số đầu tiên luôn là self. Chỉ có thể định nghĩa một constructor trong class
 - Constructor được sử dụng để tạo ra một đối tượng.
 - Constructor gán các giá trị từ tham số vào các thuộc tính của đối tượng sẽ được tạo ra

➤ Khởi tạo đối tượng từ lớp

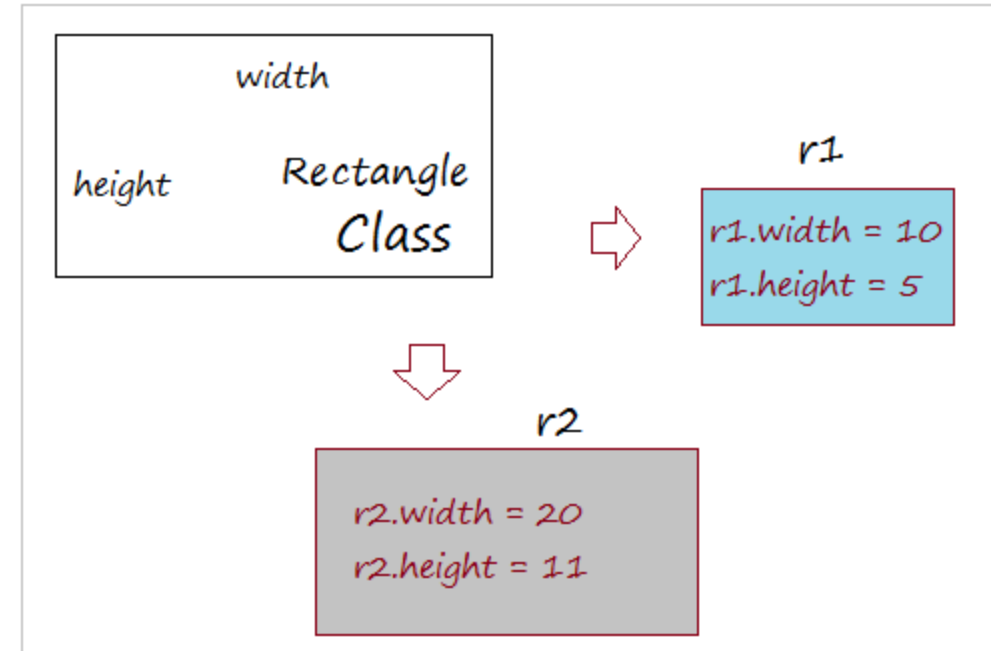
Sau khi đã khai báo được class trong Python rồi, thì để khởi tạo các đối tượng sử dụng cú pháp sau:

```
variableName = className()
```

Trong đó:

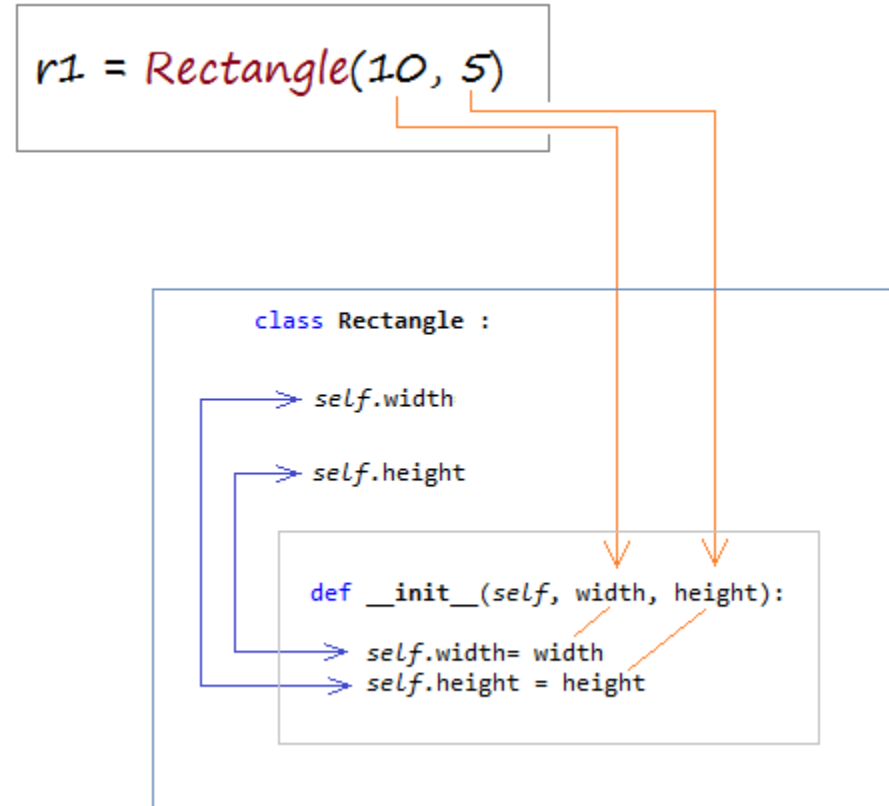
- **variableName** là tên đối tượng.
- **className** là class muốn khởi tạo.

```
1  #Ví dụ: Tạo 2 đối tượng r1, r2 từ class Rectangle
2  r1 = Rectangle(10,5)
3
4  r2 = Rectangle(20,11)
```



➤ Điều gì xảy ra khi khởi tạo một đối tượng

Khi tạo một đối tượng của lớp **Rectangle**, phương thức khởi tạo (constructor) của class đó sẽ được gọi để tạo một đối tượng, và các thuộc tính của đối tượng sẽ được gán giá trị từ tham số



- Sau khi đã khởi tạo được đối tượng sẽ có thể truy cập được các thuộc tính và phương thức trong class đó.
- Bằng cách sử dụng dấu `.` theo cú pháp sau:

```
# truy cập den thuoc tinh  
object.propertyName  
  
#truy cap den phuong thuc  
object.methodName()
```

Trong đó:

- `object` là biến thể hiện lại object.
- `propertyName` là tên thuộc tính muốn truy xuất.
- `methodName` là tên phương thức muốn truy xuất.

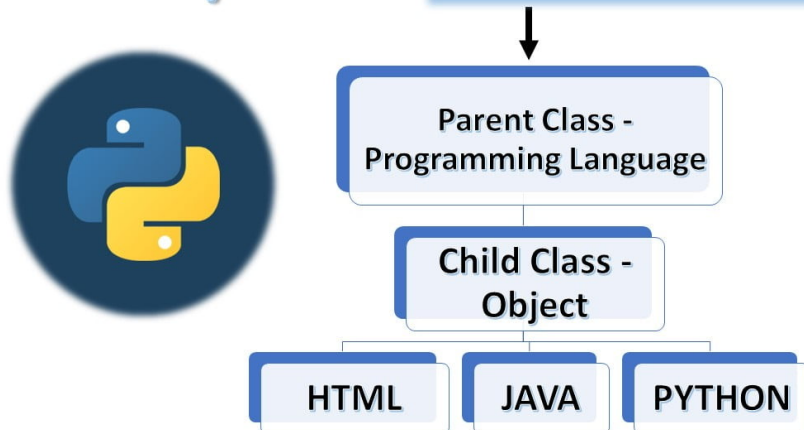
➤ **Ví dụ:** sẽ truy xuất đến các thuộc tính và phương thức trong class Rectangle

```
1  #Lấy thuộc tính width, height của đối tượng rec1
2  x = r1.width
3  y = r1.height
4  print('----Thuộc tính-----')
5  print('1. Thuộc tính Chiều rộng: ', x)
6  print('2. Thuộc tính Chiều dài: ', y)
7  #Gọi phương thức getArea, getPerimeter của đối tượng rec1
8  dt = r1.getArea()
9  cv = r1.getPerimeter()
10 print('-----Phương thức-----')
11 print('1. Phương thức tính Diện tích:', dt)
12 print('2. Phương thức tính Chu vi:', cv)
```

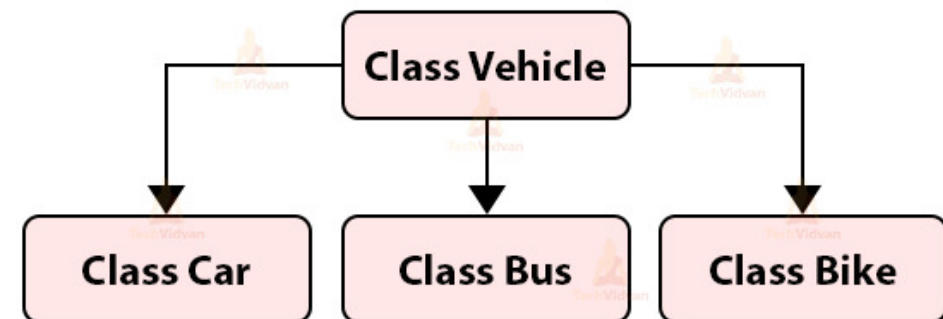
```
----Thuộc tính-----
1. Thuộc tính Chiều rộng:  10
2. Thuộc tính Chiều dài:   5
-----Phương thức-----
1. Phương thức tính Diện tích: 50
2. Phương thức tính Chu vi: 30
```

- Việc thực hiện thao tác thừa kế là đơn giản và hiệu quả, trong đó cho phép tạo ra một lớp mới từ một lớp có sẵn.
- Lớp mới dẫn xuất từ lớp có sẵn có thể tái sử dụng các biến và phương thức của nó mà không cần phải viết lại hoặc sửa lại đoạn mã.
- Khi thực hiện thừa kế, lớp được dẫn xuất từ lớp khác được gọi là phân lớp, lớp dẫn xuất, **lớp con**; Lớp mà từ đó phân lớp được dẫn xuất thì được gọi là siêu lớp, lớp cơ sở, **lớp cha**

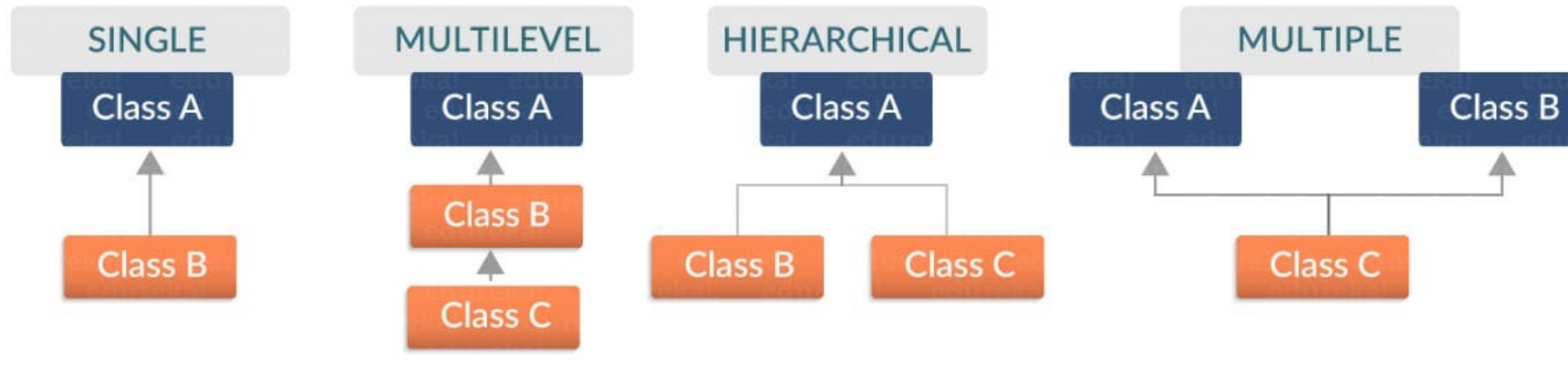
Python Inheritance



Relationship Between Classes



Types Of Inheritance



- **Đơn thừa kế (Single):** Xảy ra khi một lớp con thừa kế chỉ một lớp cha
- **Thừa kế nhiều mức (Multilevel):** Khi một lớp con dẫn xuất từ một lớp cha và bản thân lớp cha lại là con của một lớp khác.
- **Thừa kế phân cấp (Hierarchical):** Loại thừa kế này xảy ra khi một lớp cha có nhiều hơn một lớp con ở những mức khác nhau.
- **Đa thừa kế (Multiple):** Xảy ra khi một lớp con dẫn xuất từ nhiều hơn một lớp cha

- **Ví dụ:** Tạo lớp Square thừa kế từ lớp Rectangle (Đơn thừa kế)

```
#Khai báo lớp Square thừa kế từ lớp Rectangle
class Square(Rectangle):
    #dùng super() để gọi đến hàm tạo của lớp cha
    def __init__(self, width = 15, color='red'):
        super().__init__(width, width)
        self.color=color

#Thêm phương thức draw() để vẽ hình vuông theo width và color
    def draw(self):
        x = self.width
        c = self.color
        fig, ax = plt.subplots(figsize=(4,4))
        square = plt.Rectangle((0, 0), x, x, color=c)
        ax.add_patch(square)
        automin, automax = plt.xlim()
        plt.xlim(0-10, x+10)
        automin, automax = plt.ylim()
        plt.ylim(0-10, x+10)
        plt.grid(True)
        plt.show()
```

```
1 #Khái báo đối tượng a thuộc Lớp Square
2 a = Square(30, 'yellow')
```

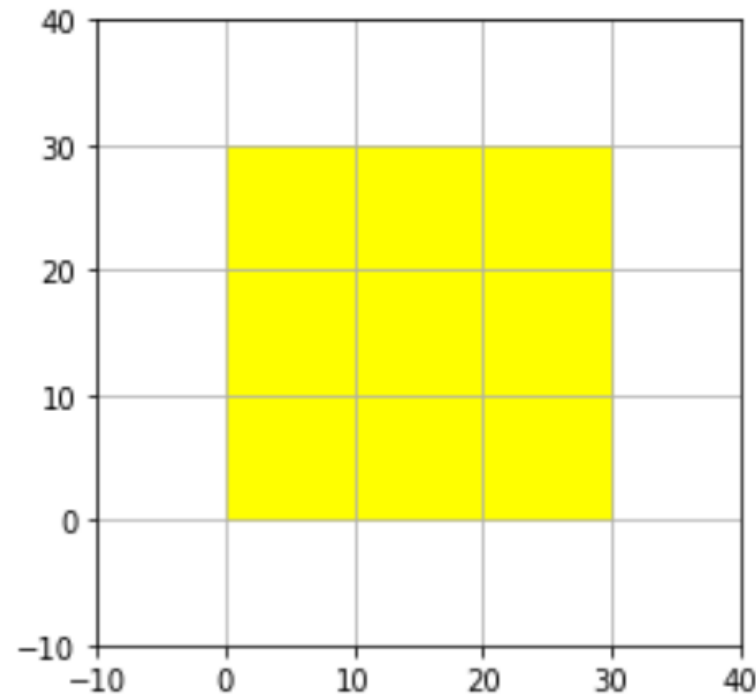
```
1 #Tái sử dụng các thuộc tính của Lớp Rectangle
2 print(a.width)
3 print(a.height)
4
5 #Bổ sung thêm thuộc tính mới cho Lớp Square
6 print(a.color)
```

```
30
30
yellow
```

```
1 #Tái sử dụng các phương thức của Lớp Rectangle
2 print('Diện tích hình vuông:', a.getArea())
3 print('Chu vi hình vuông:', a.getPerimeter())
```

```
Diện tích hình vuông: 900
Chu vi hình vuông: 120
```

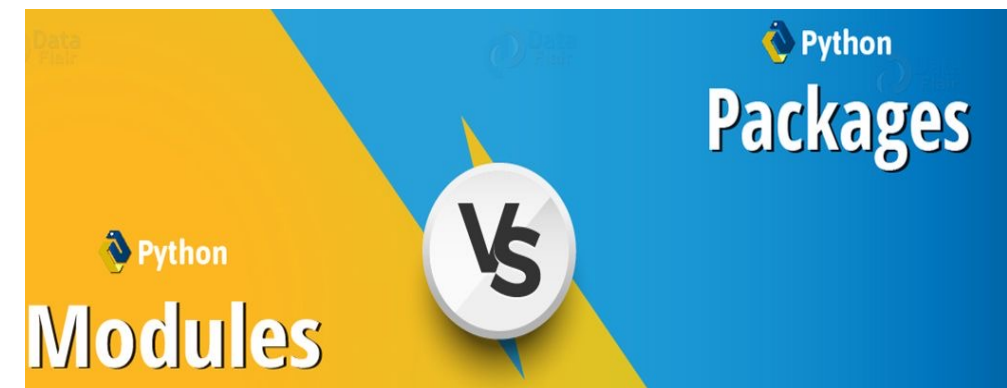
```
1 #Sử dụng phương thức mới draw của Lớp Square
2 a.draw()
```



Thực hành

3. Module và Package

- Module hiểu ngắn gọn là một file python mà trong đó chứa các khai báo và định nghĩa về hàm số và biến. Các chương trình python sẽ được thiết kế sao cho nội dung được chia nhỏ về các files module để dễ dàng quản lý. Chúng ta có thể dễ dàng import lại các khai báo và định nghĩa từ module này sang module khác.
- Một package sẽ bao gồm nhiều file module



```
1 #gọi package để sử dụng
2 import math
3
4 #Liệt kê tất cả các phương thức, thuộc tính của package math
5 print (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', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

- Khi chương trình lớn hơn, bạn nên chia nhỏ nó thành các modules khác nhau.
- Module là một file chứa các định nghĩa và câu lệnh Python.
- Module trong Python có phần mở rộng .py.
- Các định nghĩa bên trong một module có thể được sử dụng trong module khác hoặc trình thông dịch Python. Để làm điều này chúng ta sử dụng từ khóa import

```
import math
```

```
import math  
print(math.pi)
```

```
>>> from math import pi  
>>> pi 3.141592653589793
```

- Ngoài sử dụng các module có sẵn trong python ra thì chúng ta cũng có thể viết ra các module của riêng mình và import nó khi muốn sử dụng.
- Ví dụ
 - Tạo file mathplus.py có nội dung sau:

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

- Tạo file main.py sử dụng module vừa viết:

```
import mathplus  
  
print(mathplus.get_sum(4,7))  
# Kết quả: 11
```

- Định danh cho modules:
 - Trong Python, có thể gán định danh mới cho modules khi import chúng bằng cách sử dụng keyword as.

```
import modules as newname  
# hoặc đối với from import  
from modules import something as newname
```

4. Làm việc với tập tin trong Python

1. File là gì?

- File hay còn gọi là tệp, tập tin. File là tập hợp của các thông tin được đặt tên và lưu trữ trên bộ nhớ máy tính như đĩa cứng, đĩa mềm, CD, DVD,...
- Khi muốn **đọc** hoặc **ghi file**, chúng ta cần phải **mở file** trước. **Khi hoàn thành**, file cần **phải được đóng** lại để các tài nguyên gắn với file được giải phóng.
- Do đó, trong Python, một thao tác với file diễn ra theo thứ tự sau:
 - ✓ **Mở tệp tin**
 - ✓ **Đọc hoặc ghi**
 - ✓ **Đóng tệp**

2. Mở File

Để mở file trong Python chúng ta sử dụng hàm **open** với cú pháp như sau:

```
open(filePath, mode, buffer)
```

Trong đó:

- **filePath** là đường dẫn đến địa chỉ của file.
- **mode** là thông số thiết lập chế độ chúng ta mở file được cấp những quyền gì?
Mặc định mode sẽ bằng **r** (xem các mode ở dưới).
- **buffer** là thông số đệm cho file mặc định thì nó sẽ là 0.

Làm việc với tập tin (File)

Các chế độ mode

Mode	Chú thích
r	Chế độ chỉ được phép đọc.
w	Chế độ ghi file, nếu như file không tồn tại thì nó sẽ tạo mới file và ghi nội dung, còn nếu như file đã tồn tại nó sẽ ghi đè nội dung lên file cũ.
a	Mở file trong chế độ ghi tiếp. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung, và nếu như file chưa tồn tại thì nó sẽ tạo một file mới và ghi nội dung vào đó.
r+	Mở file cho cả đọc và ghi
w+	Mở file trong chế độ đọc và ghi đè lên file hiện có nếu file tồn tại, hoặc tạo ra file mới nếu chưa tồn tại
a+	Mở file trong chế độ đọc và ghi tiếp nội dung, còn lại cơ chế giống chế độ a.
x	Tạo file mới để ghi, báo lỗi nếu file đã tồn tại

2. Mở File

➤ Ví dụ:

```
f=open("test.txt") #mở file mode 'r' hoặc 'rt' để đọc
```

```
f=open("test.txt",'w') #mở file mode 'w' để ghi
```

```
import csv  
f=open('data.csv','rt') #mở file mode 'r' hoặc 'rt' để đọc file csv
```

3. Đóng File

- Việc đóng file được xây dựng trong Python bằng hàm `close()` với cú pháp như sau:

```
fileObject.close()
```

- Trong đó, `fileObject` là đối tượng mà chúng ta thu được khi sử dụng hàm `open()`.

4. Đọc file

- Sau khi đã mở được file ra rồi, để đọc được file thì chúng ta sử dụng phương thức **read** với cú pháp:

```
fileObject.read(length);
```

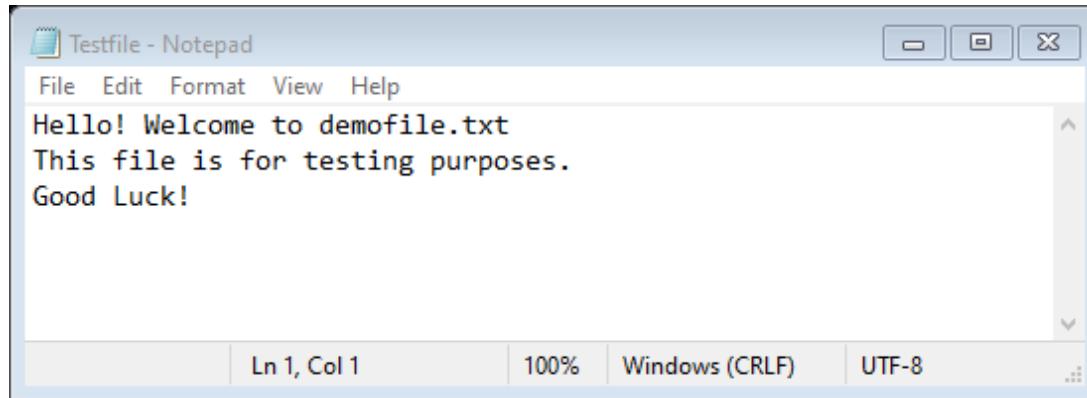
Trong đó:

- **fileObject** là đối tượng mà chúng ta thu được khi sử dụng hàm **open()**.
- **length** là dung lượng của dữ liệu mà chúng ta muốn đọc, nếu để trống tham số này thì nó sẽ đọc hết file hoặc nếu file lớn quá thì nó sẽ đọc đến khi giới hạn của bộ nhớ cho phép.

Làm việc với tập tin (File)

4. Đọc file

➤ Ví dụ:



```
1 #Mở file để đọc dữ liệu
2 f = open('Testfile.txt')
3
4 #Đọc nội dung của file vào biến st
5 st = f.read()
6
7 print('Nội dung file:')
8 print(st)
```

Nội dung file:
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!

```
1 #Mở file để đọc dữ liệu
2 f = open('Testfile.txt','r')
3
4 #Đọc 10 ký tự đầu tiên của file
5 st1 = f.read(10)
6
7 print(st1, ' -- Số ký tự là: ', len(st1))
```

Hello! Wel -- Số ký tự là: 10

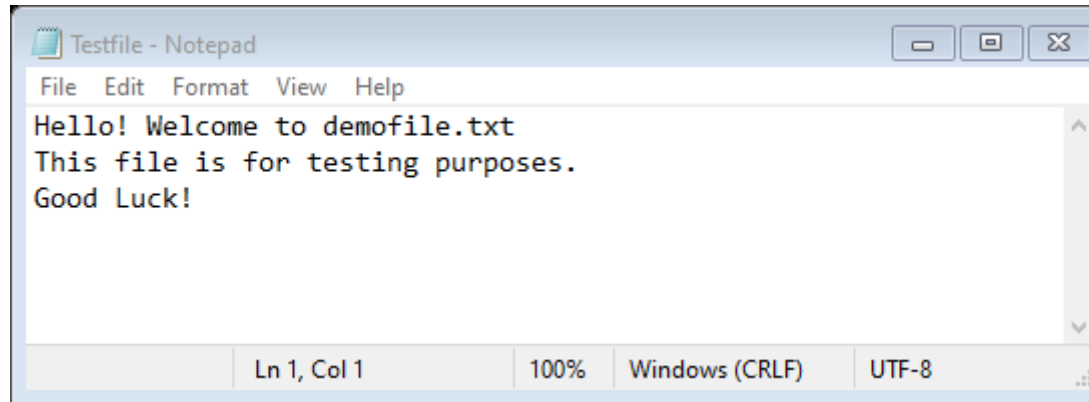
```
1 #Mở file đọc dữ liệu với with:
2 with open('Testfile.txt') as f:
3     lines = f.readlines()
4
5 print(lines)
6 f.close()
```

['Hello! Welcome to demofile.txt\n', 'This

Làm việc với tập tin (File)

4. Đọc file

➤ Ví dụ:



```
1 #Mở file để đọc dữ liệu
2 f = open('Testfile.txt')
3
4 #Đọc từng dòng dữ liệu của file
5 print(f.readline())
6 print(f.readline())
7
8 f.close() #Đóng file dữ liệu
```

Hello! Welcome to demofile.txt

This file is for testing purposes.

```
1 #Mở file để đọc dữ liệu
2 f = open("Testfile.txt", "r")
3
4 #đọc tất cả các dòng của file
5 for x in f:
6     print(x)
7
8 f.close() #Đóng file dữ liệu
```

Hello! Welcome to demofile.txt

This file is for testing purposes.

Good Luck!

5. Ghi file

- Để ghi được file thì bạn phải chắc chắn là đang mở file ở các chế độ cho phép ghi. Và sử dụng phương thức **write** với cú pháp sau:

```
fileObject.write(data)
```

Trong đó:

- fileObject** là đối tượng mà chúng ta thu được khi sử dụng hàm **open()**.
- data** là dữ liệu mà chúng ta muốn ghi vào trong file.

➤ Ví dụ:

```
1  #Mở file với chế độ ghi đè (w):  
2  #nếu như file không tồn tại thì nó sẽ tạo mới file và ghi nội dung,  
3  #còn nếu như file đã tồn tại nó sẽ ghi đè nội dung lên file cũ.  
4  f1 = open('Ghifile.txt', 'w')  
5  
6  #Dữ liệu muốn ghi vào file  
7  st = 'Welcome to Python for Analysis!'  
8  
9  f1.write(st) #Ghi dữ liệu vào file  
10 f1.close() #Đóng file
```

5. Ghi file

➤ Ví dụ:

```
1  #Mở file với chế độ ghi tiếp (a):
2  # Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung,
3  # và nếu như file chưa tồn tại thì nó sẽ tạo một file mới và ghi nội dung vào đó.
4  f1 = open('Ghifile.txt', 'a+')
5
6  #Dữ liệu muốn ghi vào file
7  st = 'This is new line.....'
8
9  f1.write(st) #Ghi tiếp dữ liệu vào file
10 f1.close() #Đóng file
11
12 #open and read the file after the appending:
13 f = open("Ghifile.txt", "r")
14 print(f.read())
```

Welcome to Python for Analysis!This is new line.....

➤ Các thuộc tính trong file.

Thuộc tính	Chú thích
<code>file.name</code>	Trả về tên của file đang được mở.
<code>file.mode</code>	Trả về chế độ mode của file đang được mở.
<code>file.closed</code>	Trả về true nếu file đã được đóng, và false nếu file chưa đóng.

➤ Ví dụ:

In ra thông số của file Ghifile.txt ở trên

```
1 #Lấy các thông số của file
2 f2 = open('Ghifile.txt')
3
4 print('1.Tên file:',f2.name)
5 print('2.Chế độ mở file:',f2.mode)
6 print('3.Trạng thái đóng file:',f2.closed)
```

```
1.Tên file: Ghifile.txt
2.Chế độ mở file: r
3.Trạng thái đóng file: False
```


Bài 1: Ghi dữ liệu vào File “data.txt”

```
# Mở file để ghi
fo = open("data.txt", "w")
# Ghi dữ liệu lên file
fo.write("Tobe or not tobe. \n Nghi lon de thanh cong ! \n");
# Close opened file
fo.close()
print("Ghi file thanh cong !")
```

Bài 2: Đọc và ghi dữ liệu từ một File

```
obj=open("test.txt","w")
obj.write("Chao mung cac ban den voi khoa CNTT")
obj.close()
obj1=open("test.txt","r")
s=obj1.read()
print (s)
obj1.close()
obj2=open("test.txt","r")
s1=obj2.read(20)
print (s1)
obj2.close()
```

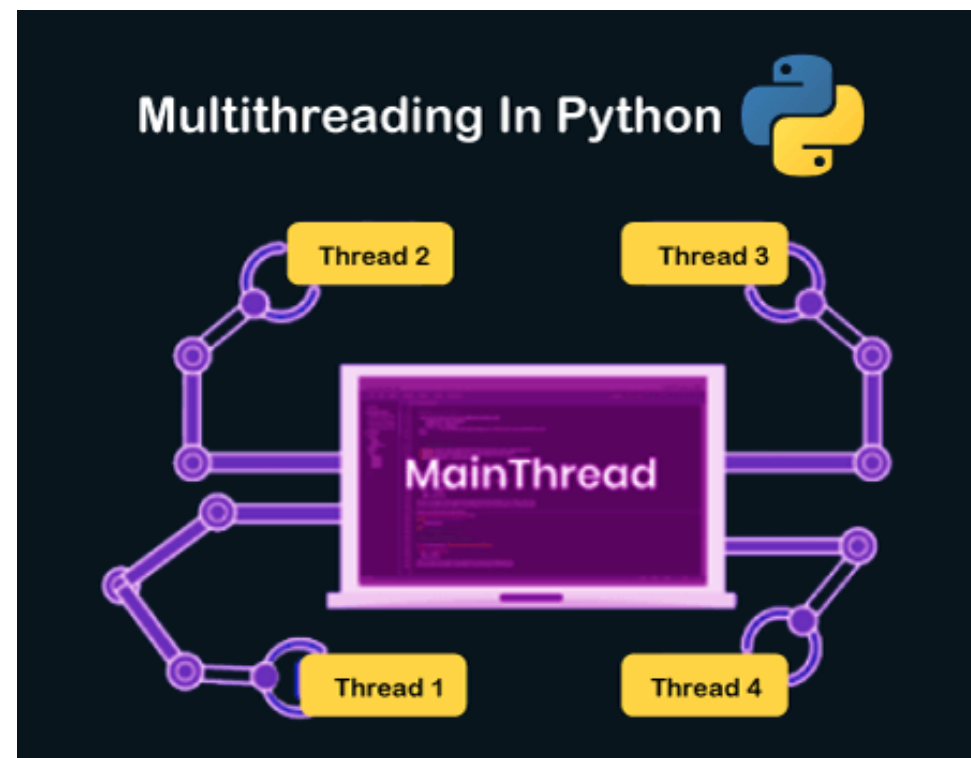
Thực hành

5. Multithreading, Multiprocessing, Functools

Đa luồng (Multithreading)?

- Một chương trình đa luồng chứa hai hoặc nhiều phần mà có thể chạy đồng thời và mỗi phần có thể xử lý tác vụ khác nhau tại cùng một thời điểm được gọi là Đa luồng. → Giúp sử dụng tốt nhất các tài nguyên sẵn có của máy tính.

- Python cung cấp thread Module và threading Module để bạn có thể bắt đầu một thread mới cũng như một số tác vụ khác trong khi lập trình đa luồng.
- Mỗi một Thread đều có vòng đời chung là bắt đầu, chạy và kết thúc.
- Một Thread có thể bị ngắt (interrupt), hoặc tạm thời bị dừng (sleeping) trong khi các Thread khác đang chạy



<https://docs.python.org/3/library/threading.html#>

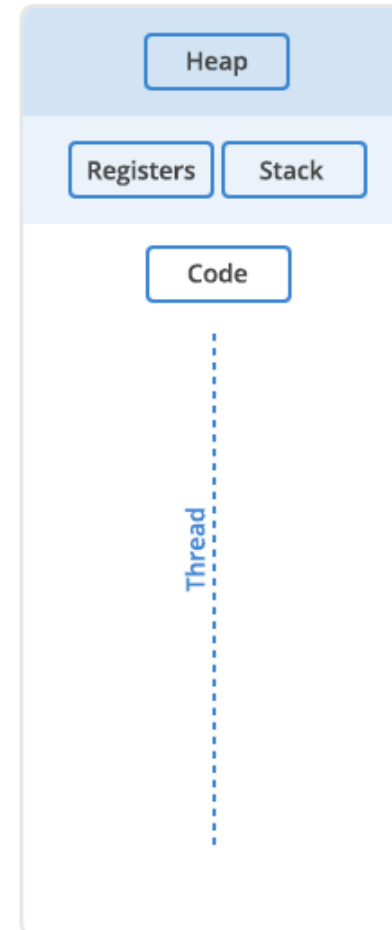
Multithreading

So sánh đơn luồng và đa luồng

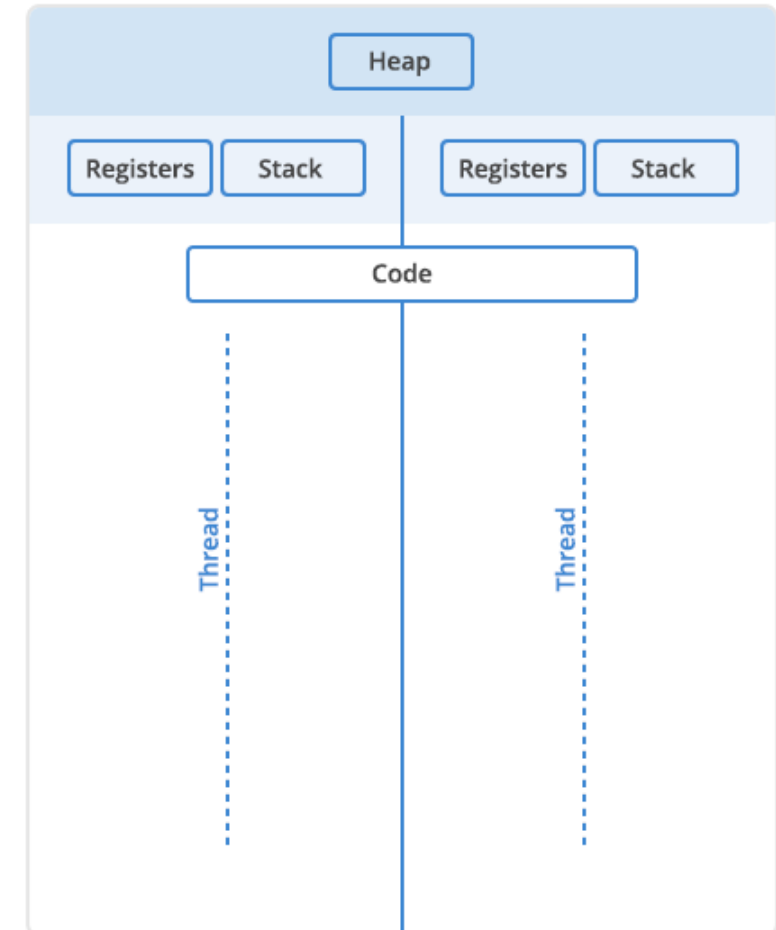
- Xây dựng hàm tính X bình phương và lập phương của các số:

```
1 #Hàm tính bình phương của x:
2 def cal_square(numbers):
3     print("calculate square number")
4     for n in numbers:
5         time.sleep(0.2)
6         print ('square:', n*n)
7
8 #Hàm tính lập phương của x:
9 def cal_cube(numbers):
10    print("calculate cube number")
11    for n in numbers:
12        time.sleep(0.2)
13        print ('cube:', n*n*n)
```

Single Thread



Multi Threaded



Multithreading

So sánh đơn luồng và đa luồng

```
1 #Thời gian thực hiện với đơn luồng (tuần tự)
2 import time
3 arr = [15,27,49,68]
4 t = time.time()
5 cal_square(arr) #Thực hiện tính arr bình phương
6 cal_cube(arr)   #Thực hiện tính arr lập phương
7 print ("done in ", time.time()- t)
```

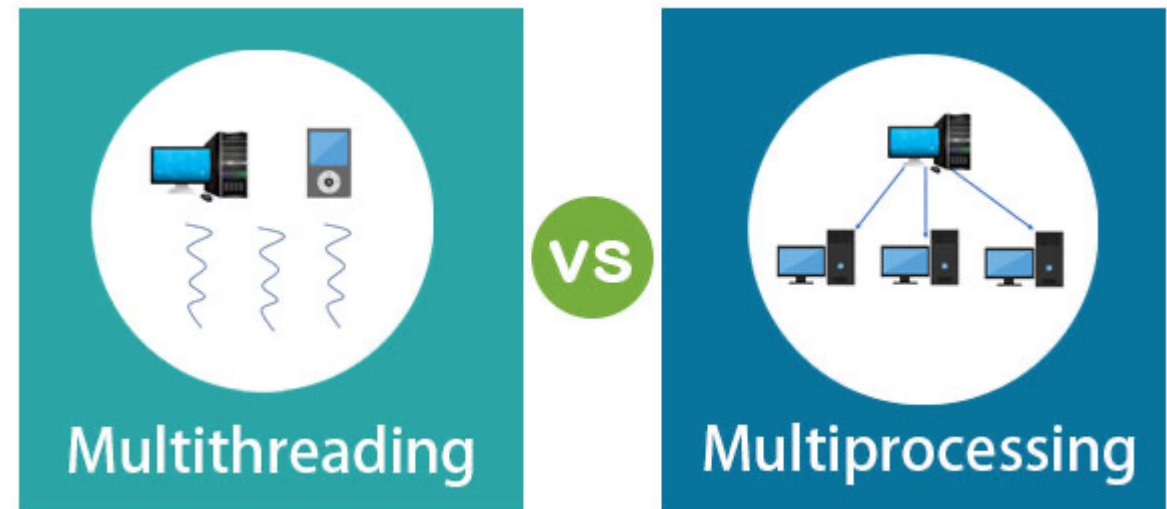
```
calculate square number
square: 225
square: 729
square: 2401
square: 4624
calculate cube number
cube: 3375
cube: 19683
cube: 117649
cube: 314432
done in  1.682037115097046
```

```
1 #Thời gian thực hiện với 2 luồng
2 from threading import Thread
3 import threading
4 import time
5 arr = [15,27,49,68]
6 try:
7     t = time.time()
8     #Tạo 2 luồng thread
9     t1 = threading.Thread(target=cal_square, args=(arr,))
10    t2 = threading.Thread(target=cal_cube, args=(arr,))
11    t1.start() #Bắt đầu thread 1
12    t2.start() #Bắt đầu thread 2
13    t1.join()  #Chờ tới khi thread 1 hoàn thành
14    t2.join()  #Chờ tới khi thread 2 hoàn thành
15    print ("done in ", time.time()- t)
16 except:
17    print ("error")
```

```
calculate square number
calculate cube number
square: 225
cube: 3375
square: 729
cube: 19683
square: 2401
cube: 117649
square: 4624
cube: 314432
done in  0.8385195732116699
```

Multiprocessing

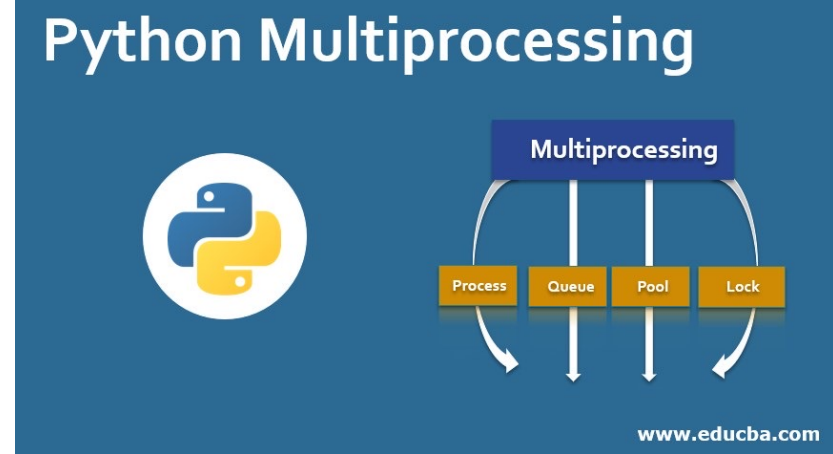
- Xử lý đa tiến trình (Multiprocessing) là khả năng của một hệ thống hỗ trợ nhiều bộ vi xử lý cùng một lúc. Các ứng dụng của hệ thống đa xử lý được chia thành nhiều tiến trình nhỏ và chạy độc lập cùng nhau (xử lý song song) → Cải thiện hiệu suất của hệ thống



<https://laptrinhx.com/multithreading-vs-multiprocessing-1494474039/>

Multiprocessing

- Thư viện Multiprocessing trong Python là một module hỗ trợ lập trình viên có thể phân chia công việc theo nhiều tiến trình. Bằng cách thông qua những phương thức (API) mà module cung cấp sẵn, chúng ta có thể quản lý được các task một cách dễ dàng.



```
1 #Sử dụng thư viện multiprocessing kiểm tra số lượng CPU
2 import multiprocessing
3 print("Số lượng CPU: ", multiprocessing.cpu_count())
```

Số lượng CPU: 4

Multiprocessing

Ví dụ về xử lý đa tiến trình với thư viện Multiprocessing của Python:

```
1 #Hàm tính bình phương của x:
2 def cal_square(numbers):
3     print("calculate square number")
4     for n in numbers:
5         time.sleep(0.2)
6         print ('square:', n*n)
7
8 #Hàm tính lập phương của x:
9 def cal_cube(numbers):
10    print("calculate cube number")
11    for n in numbers:
12        time.sleep(0.2)
13        print ('cube:', n*n*n)
```

```
1 #Tạo 2 tiến trình chạy song song với multiprocessing
2 import multiprocessing
3 import time
4 arr = [15,27,49,68]
5 try:
6     t = time.time()
7     #Tạo 2 tiến trình process
8     p1 = multiprocessing.Process(target=cal_square, args=(arr,))
9     p2 = multiprocessing.Process(target=cal_cube, args=(arr,))
10    p1.start()    #Bắt đầu process 1
11    p2.start()    #Bắt đầu process 2
12    p1.join()     #Chờ tới khi process 1 hoàn thành
13    p2.join()     #Chờ tới khi process 1 hoàn thành
14    print ("done in ", time.time()- t)
15 except:
16    print ("error")
```

```
calculate square number
calculate cube number
square: 225
cube: 3375
square: 729
cube: 19683
square: 2401
cube: 117649
square: 4624
cube: 314432
done in  0.8370239734649658
```

Tham khảo:

<https://phamdinhhkhanh.github.io/2020/11/30/ParallelComputingPython.html>

<https://docs.python.org/3.1/library/multiprocessing.html>

Functools là một thư viện của Python, cung cấp các tính năng hữu ích giúp làm việc với các hàm bậc cao dễ dàng hơn.

Molude này bao gồm các chức năng chính:

```
@functools.cache()
```

```
@functools.cached_property()
```

```
@functools.lru_cache()
```

```
@functools.lru_cache()
```

```
@functools.total_ordering
```

```
@functools.reduce()
```

```
@functools.singledispatch
```

```
@functools.wraps()
```



Tham khảo:

<https://docs.python.org/3/library/functools.html>

Q & A
Thank you!