# THỰC HÀNH BUỔI 2
# MÔN: CƠ SỞ TRÍ TUỆ NHÂN TẠO

## 1. Cú pháp của hàm Python

```
def ten_ham(các tham số/đối số):
 """Chuỗi văn bản để mô tả cho hàm (docstring)"""
 Các câu lệnh
```

Về cơ bản, một định nghĩa hàm Python sẽ bao gồm các thành phần sau:

1. Từ khóa def: Đánh dáu sự bắt đầu của tiêu đề hàm.
2. ten_ham: Là định danh duy nhất dành cho hàm. Việc đặt tên hàm phải tuân thủ theo quy tắc viết tên và định danh trong Python.
3. Các tham số/đối số: Chúng ta truyền giá trị cho hàm thông qua các tham số này. Chúng là tùy chọn.
4. Dấu hai chấm (:): Đánh dấu sự kết thúc của tiêu đề hàm.
5. docstring: Chuỗi văn bản tùy chọn để mô tả chức năng của hàm.
6. Các câu lệnh: Một hoặc nhiều lệnh Python hợp lệ tạo thành khối lệnh. Các lệnh này phải có cùng một mức canh lề đầu dòng (thường là 4 khoảng trắng).
7. Lệnh return: Lệnh này là tùy chọn, dùng khi cần trả về giá trị từ hàm.

Ví dụ:

```
def gia_tri_tuyet_doi(so):
    """Hàm này trả về giá trị tuyệt đối
    của một số nhập vào"""
    if so >= 0:
    return so
    else:
        return -so
# Đầu ra: 5
print(gia_tri_tuyet_doi(5))
# Đầu ra: 8
print(gia_tri_tuyet_doi(-8))
# Đầu ra: Giá trị tuyệt đối của số nhập vào
num=int(input("Nhập số cần lấy giá trị tuyệt đối: "))
print (gia_tri_tuyet_doi(num))
```

Khi chạy code trên, ta được kết quả như sau:

```
5
8
Nhập số cần lấy giá trị tuyệt đối: -7
7
```

# 2. Phạm vi và thời gian tồn tại của các biến

Thời gian tồn tại của biến là khoảng thời gian mà biến đó xuất hiện trong bộ nhớ. Khi hàm được thực thi thì biến sẽ tồn tại. Biến bị hủy khi chúng ta thoát khỏi hàm. Hàm không nhớ giá trị của biến trong những lần gọi hàm trước đó.

```
def ham_in():
 x = 15
    print("Giá trị bên trong hàm:",x)
x = 30
ham_in()
print("Giá trị bên ngoài hàm:",x)
```

```
Giá trị bên trong hàm: 15
Giá trị bên ngoài hàm: 30
```

# 3. Các hàm Python tích hợp sẵn

| Hàm | Mô tả |
|---|---|
| abs() | Trả về giá trị tuyệt đối của một số |
| all() | Trả về True khi tất cả các phần tử trong iterable là đúng |
| any() | Kiểm tra bất kỳ phần tử nào của iterable là True |
| ascii() | Tả về string chứa đại diện (representation) có thể in |
| bin() | Chuyển đổi số nguyên sang chuỗi nhị phân |
| bool() | Chuyển một giá trị sang Boolean |
| bytearray() | Trả về mảng kích thước byte được cấp |
| bytes() | Trả về đối tượng byte không đổi |
| callable() | Kiểm tra xem đối tượng có thể gọi hay không |
| chr() | Trả về một ký tự (một chuỗi) từ Integer |
| classmethod() | Trả về một class method cho hàm |
| compile() | Trả về đối tượng code Python |
| complex() | Tạo một số phức |
| delattr() | Xóa thuộc tính khỏi đối tượng |
| dict() | Tạo Dictionary |

| Hàm | Mô tả |
|---|---|
| dir() | Trả lại thuộc tính của đối tượng |
| divmod() | Trả về một Tuple của Quotient và Remainder |
| enumerate() | Trả về đối tượng kê khai |
| eval() | Chạy code Python trong chương trình |
| exec() | Thực thi chương trình được tạo động |
| filter() | Xây dựng iterator từ các phần tử True |
| float() | Trả về số thập phân từ số, chuỗi |
| format() | Trả về representation được định dạng của giá trị |
| frozenset() | Trả về đối tượng frozenset không thay đổi |
| getattr() | Trả về giá trị thuộc tính được đặt tên của đối tượng |
| globals() | Trả về dictionary của bảng sumbol toàn cục hiện tại |
| hasattr() | Trả về đối tượng dù có thuộc tính được đặt tên hay không |
| hash() | Trả về giá trị hash của đối tượng |
| help() | Gọi Help System được tích hợp sẵn |
| hex() | Chuyển Integer thành Hexadecimal |
| id() | Trả về định danh của đối tượng |
| input() | Đọc và trả về chuỗi trong một dòng |
| int() | Trả về số nguyên từ số hoặc chuỗi |
| isinstance() | Kiểm tra xem đối tượng có là Instance của Class không |
| issubclass() | Kiểm tra xem đối tượng có là Subclass của Class không |
| iter() | Trả về iterator cho đối tượng |
| len() | Trả về độ dài của đối tượng |
| list() | Tạo list trong Python |
| locals() | Trả về dictionary của bảng sumbol cục bộ hiện tại |
| map() | Áp dụng hàm và trả về một list |
| max() | Trả về phần tử lớn nhất |
| memoryview() | Trả về chế độ xem bộ nhớ của đối số |

| Hàm | Mô tả |
|---|---|
| min() | Trả về phần tử nhỏ nhất |
| next() | Trích xuất phần tử tiếp theo từ Iterator |
| object() | Tạo một đối tượng không có tính năng (Featureless Object) |
| oct() | Chuyển số nguyên sang bát phân |
| open() | Trả về đối tượng File |
| ord() | Trả về mã Unicode code cho ký tự Unicode |
| pow() | Trả về x mũ y |
| print() | In đối tượng được cung cấp |
| property() | Trả về thuộc tính property |
| range() | Trả về chuỗi số nguyên từ số bắt đầu đến số kết thúc |
| repr() | Trả về representation có thể in của đối tượng |
| reversed() | Trả về iterator đảo ngược của một dãy |
| round() | Làm tròn số thập phân |
| set() | Tạo một set các phần tử mới |
| setattr() | Đặt giá trị cho một thuộc tính của đối tượng |
| slice() | Cắt đối tượng được chỉ định bằng range() |
| sorted() | Trả về list được sắp xếp |
| staticmethod() | Tạo static method từ một hàm |
| str() | Trả về một representation không chính thức của một đối tượng |
| sum() | Thêm một mục vào Iterable |
| super() | Cho phép tham chiếu đến Parent Class bằng super |
| tuple() | Tạo một Tuple |
| type() | Trả về kiểu đối tượng |
| vars() | Trả về thuộc tính __dict__ của class |
| zip() | Trả về Iterator của Tuple |
| __import__() | Hàm nâng cao, được gọi bằng import |

# 4. Hàm Python do người dùng tự định nghĩa

```
def ten_ham(DoiSo1,DoiSo2,...,DoiSoN)
   khối lệnh của hàm
```

Ví dụ:

```
def them_so(a,b):
 tong = a + b
 return tong

so1 = 5
so2 = 6
so3 = int(input("Nhập một số: "))
so4 = int(input("Nhập một số nữa: "))

print("Tổng hai số đầu là: ", them_so(so1, so2))

print ("Tổng của hai số sau là: ", them_so(so3, so4))
```

```
Nhập một số: 8
Nhập một số nữa: 10
Tổng hai số đầu là: 11
Tổng của hai số sau là: 18
```

# Hàm đệ quy trong Python

Ví dụ, 5 giai thừa (được viết là 5!) là 1*2*3*4*5= 120.

```
def giaithua(n):
 """Đây là hàm tính giai thừa của
    một số nguyên by Quantrimang.com"""
 if n == 1:
 return 1
 else:
       return (n * giaithua(n-1))

num = 5
num1 = int(input("Nhập số cần tính giai thừa: "))
print("Giai thừa của", num, "là", giaithua(num))
print("Giai thừa của", num1, "là", giaithua(num1))
```

# 5. Biến toàn cục (global), biến cục bộ (local), biến nonlocal trong Python

```
x = 5

def vidu():
 x = 10
 print("Biến x cục bộ:", x)

vidu()
print("Biến x toàn cục:", x)
```

```
Biến x cục bộ: 10
Biến x toàn cục: 5
```

# Từ khóa global trong Python

- Khi chúng ta tạo biến trong một hàm, nó mặc định là biến cục bộ.
- Khi chúng ta định nghĩa một biến bên ngoài hàm, nó mặc định là biến toàn cục. Bạn không cần phải sử dụng từ khóa global.
- Chúng ta sử dụng từ khóa global để đọc và viết biến toàn cục trong một hàm.
- Sử dụng từ khóa global bên ngoài một hàm thì không có tác dụng gì cả.

```
def ham1():
 x = 20

 def ham2():
 global x
 x = 25

 print("Trước khi gọi ham2: ", x)
 print("Đang gọi ham2")
 ham2()
 print("Sau khi gọi ham2: ", x)

ham1()
print("x trong main: ", x)
```

```
Trước khi gọi ham2: 20
Đang gọi ham2
Sau khi gọi ham2: 20
x trong main: 25
```

# Biến nonlocal trong Python

Trong Python, biến nonlocal được sử dụng trong hàm lồng nhau nơi mà phạm vi cục bộ không được định nghĩa. Nói dễ hiểu thì biến nonlocal không phải biến local, không phải biến global, bạn khai báo một biến là nonlocal khi muốn sử dụng nó ở phạm vi rộng hơn local, nhưng chưa đến mức global.

```
def ham_ngoai():
    x = "Biến cục bộ"

    def ham_trong():
        nonlocal x
        x = "Biến nonlocal"
        print("Bên trong:", x)

    ham_trong()
    print("Bên ngoài:", x)

hamngoai()
```

```
Bên trong: Biến nonlocal
Bên ngoài: Biến nonlocal
```

# 6. Làm việc với file

# Mở/ Đóng File trong Python

```
f = open("test.txt") # mở file cùng thư mục với file hiện tại
f = open("C:/Python33/README.txt") # mở file ở thư mục khác, đường dẫn đầy đủ
```

```
f = open("test.txt") # mở file mode 'r' hoặc 'rt' để đọc
f = open("test.txt",'w') # mở file mode ‘w’ để ghi
f = open("img.bmp",'r+b') # mở file mode ‘r+b’ để đọc và ghi dạng nhị phân
```

```
f = open("test.txt",encoding = 'utf-8')
# thực hiện các thao tác với tệp
f.close()
```

Dưới đây là danh sách các chế độ mode khác nhau khi mở một file:

| MODE | MÔ TẢ |
|---|---|
| ‘r’ | Chế độ chỉ được phép đọc. |
| ‘r+’ | Chế độ được phép đọc và ghi |

| | |
|---|---|
| ‘rb’ | Mở file chế độ đọc cho định dạng nhị phân. Con trỏ tại phần bắt đầu của file |
| ‘rb+’<br>‘r+b’ | Mở file để đọc và ghi trong định dạng nhị phân. Con trỏ tại phần bắt đầu của file |
| ‘w’ | Mở file để ghi. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ |
| ‘w+’ | Mở file để đọc và ghi. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ |
| ‘wb’ | Mở file để ghi cho dạng nhị phân. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ |
| ‘wb+’<br>‘w+b’ | Mở file để đọc và ghi cho dạng nhị phân. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ |
| ‘a’ | Mở file chế độ ghi tiếp. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó. |
| ‘a+’ | Mở file chế độ đọc và ghi tiếp. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó. |
| ‘ab’, | Mở file chế độ ghi tiếp ở dạng nhị phân. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó. |
| ‘ab+’<br>‘a+b | Mở file chế độ đọc và ghi tiếp ở dạng nhị phân. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó. |
| ‘x’ | Mở file chế độ ghi. Tạo file độc quyền mới (exclusive creation) và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi |
| ‘x+’ | Mở file chế độ đọc và ghi. Tạo file độc quyền mới (exclusive creation) và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi |
| ‘xb’ | Mở file chế độ ghi dạng nhị phân. Tạo file độc quyền mới và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi |
| ‘xb+’<br>‘x+b’ | Mở file chế độ đọc và ghi dạng nhị phân. Tạo file độc quyền mới và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi |
| ‘b’ | Mở file ở chế độ nhị phân |
| ‘t’ | Mở file ở chế độ văn bản (mặc định) |

Để đảm bảo hơn, bạn nên sử dụng khối lệnh try...finally (finally sẽ luôn luôn được thực thi bất chấp có hay không ngoại lệ) ở đây.

```
try:
 f = open("test.txt",encoding = 'utf-8')
 # thực hiện các thao tác với tệp
finally:
 f.close()
```

# Ghi File trong Python

```
with open("test.txt",'w',encoding = 'utf-8') as f:
 f.write("Kiến thức - Kinh nghiệm - Hỏi đáp\n\n")
```

# Đọc File trong Python

**Dùng read(size)**

```
f = open("test.txt",'r',encoding = 'utf-8')
a = f.read(12) # đọc 12 kí tự đầu tiên
print('Nội dung 11 kí tự đầu là:\n', (a))
b = f.read(35) # đọc 35 kí tự tiếp theo
print('Nội dung 35 kí tự tiếp theo là:\n', (b))
c = f.read() # đọc phần còn lại
print('Nội dung phần còn lại là:\n', (c))
```

**Dùng tell() và seek()**

```
f = open("test.txt",'r',encoding = 'utf-8')
a = f.read(12) # đọc 12 kí tự đầu tiên
print('Nội dung là: \n', (a))

b = f.tell() # Kiểm tra vị trí hiện tại
print ('Vị trí hiện tại: ', (b))

f.seek(0) # Đặt lại vị trí con trỏ tại vị trí đầu file
c = f.read()
print('Nội dung mới là: \n', (c))
```

**Dùng readline()**

Phương thức này cho phép đọc từng dòng trong file:

```
f = open("test.txt",'r',encoding = 'utf-8')
a = f.readline()
print ('Nội dung dòng đầu: ', (a))
b = f.readline()
print ('Nội dung dòng 2: ', (b))
c = f.readline()
print ('Nội dung dòng 3: ', (c))
d = f.readline()
print ('Nội dung dòng 4: ', (d))
```

**Dùng readlines()**

Phương thức *readlines()* trả về toàn bộ các dòng còn lại trong file và trả về giá trị rỗng khi kết thúc file.

```
f = open("test.txt",'r',encoding = 'utf-8')
a = f.readline()
print ('Nội dung dòng đầu: ', (a))
b = f.readlines()
print ('Nội dung các dòng còn lại: \n', (b))
c = f.readlines()
print ('Nội dung các dòng còn lại: \n', (c))
```

# Một số phương thức làm việc với File trong Python

| PHƯƠNG THỨC | MÔ TẢ |
|---|---|
| close() | Đóng một file đang mở. Nó không thực thi được nếu tập tin đã bị đóng. |
| fileno() | Trả về một số nguyên mô tả file (file descriptor). |
| flush() | Xóa sạch bộ nhớ đệm của luồng file. |
| isatty() | Trả về TRUE nếu file được kết nối với một thiết bị đầu cuối. |
| read(n) | Đọc n kí tự trong file. |
| readable() | Trả về TRUE nếu file có thể đọc được. |
| readline(n=-1) | Đọc và trả về một dòng từ file. Đọc nhiều nhất n byte/ký tự nếu được chỉ định. |
| readlines(n=-1) | Đọc và trả về một danh sách các dòng từ file. Đọc nhiều nhất n byte/ký tự nếu được chỉ định. |
| seek(offset,from=SEEK_SET) | Thay đổi vị trí hiện tại bên trong file. |
| seekable() | Trả về TRUE nếu luồng hỗ trợ truy cập ngẫu nhiên. |
| tell() | Trả về vị trí hiện tại bên trong file. |
| truncate(size=None) | Cắt gọn kích cỡ file thành kích cỡ tham số size. |
| writable() | Trả về TRUE nếu file có thể ghi được. |
| write(s) | Ghi s kí tự vào trong file và trả về. |
| writelines(lines) | Ghi một danh sách các dòng và file. |

# 7. Xử lý ngoại lệ - Exception Handling trong Python

```
try:
 # khối code lệnh try
except exceptionName:
 # khối code lệnh except
```

```
try:
 f = open("test.txt",encoding = 'utf-8')
 # thực hiện các thao tác với tệp
finally:
 f.close()
```

Ví dụ:

```
mauso = input("Bạn hãy nhập giá trị mẫu số: ")
try:
 ketqua = 15/int(mauso)
 print("Kết quả là:",ketqua)
finally:
 print("Bạn đã nhập số không thể thực hiện phép tính.")
```

```
Bạn hãy nhập giá trị mẫu số: 5
Kết quả là: 3.0
Bạn đã nhập số không thể thực hiện phép tính.
```

# 8 – Lập trình hướng đối tượng trong Python

Trong Python, khái niệm về OOP tuân theo một số nguyên lý cơ bản là *tính đóng gói, tính kế thừa và tính đa hình*.

- *Tính kế thừa (Inheritance)*: cho phép một lớp (class) có thể kế thừa các thuộc tính và phương thức từ các lớp khác đã được định nghĩa.
- *Tính đóng gói (Encapsulation)*: là quy tắc yêu cầu trạng thái bên trong của một đối tượng được bảo vệ và tránh truy cập được từ code bên ngoài (tức là code bên ngoài không thể trực tiếp nhìn thấy và thay đổi trạng thái của đối tượng đó).
- *Tính đa hình (Polymorphism):* là khái niệm mà hai hoặc nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách thức khác nhau.

# Lớp (Class) và Đối tượng (Object)

Class và Object là hai khái niệm cơ bản trong lập trình hướng đối tượng.

**Đối tượng (Object)** là những thực thể tồn tại có hành vi.

Ví dụ đối tượng là một xe ô tô có tên hãng, màu sắc, loại nguyên liệu, hành vi đi, dừng, đỗ, nổ máy...

**Lớp (Class)** là một kiểu dữ liệu đặc biệt do người dùng định nghĩa, tập hợp nhiều thuộc tính đặc trưng cho mọi đối tượng được tạo ra từ lớp đó.

Thuộc tính là các giá trị của lớp. Sau này khi các đối tượng được tạo ra từ lớp, thì thuộc tính của lớp lúc này sẽ trở thành các đặc điểm của đối tượng đó.

**Phân biệt giữa Đối tượng (Object) và Lớp (Class):**

*Đối tượng (Object):* có trạng thái và hành vi.

*Lớp (Class):* có thể được định nghĩa như là một template mô tả trạng thái và hành vi mà loại đối tượng của lớp hỗ trợ. Một đối tượng là một thực thể (instance) của một lớp


**Ví dụ về Class và Object:**

```
class Car:

    # thuộc tính lớp
    loaixe = "Ô tô"

    # thuộc tính đối tượng
    def __init__(self, tenxe, mausac, nguyenlieu):
      self.tenxe = tenxe
      self.mausac = mausac
      self.nguyenlieu = nguyenlieu

# instantiate the Car class
toyota = Car("Toyota", "Đỏ", "Điện")
lamborghini = Car("Lamborghini", "Vàng", "Deisel")
porsche = Car("Porsche", "Xanh", "Gas")

# access the class attributes
print("Porsche là {}.".format(porsche.__class__.loaixe))
print("Toyota là {}.".format(toyota.__class__.loaixe))
```

```
print("Lamborghini cũng là {}.".format(lamborghini.__class__.loaixe))

# access the instance attributes
print("Xe {} có màu {}. {} là nguyên liệu vận hành.".format( toyota.tenxe, toyota.mausac, toyota.nguyenlieu))
print("Xe {} có màu {}. {} là nguyên liệu vận hành.".format( lamborghini.tenxe,
lamborghini.mausac,lamborghini.nguyenlieu))
print("Xe {} có màu {}. {} là nguyên liệu vận hành.".format( porsche.tenxe, porsche.mausac, porsche.nguyenlieu))
```

Kết quả trả về sẽ là:

```
Porsche là Ô tô.
Toyota là Ô tô.
Lamborghini cũng là Ô tô.
Xe Toyota có màu Đỏ. Điện là nguyên liệu vận hành.
Xe Lamborghini có màu Vàng. Deisel là nguyên liệu vận hành.
Xe Porsche có màu Xanh. Gas là nguyên liệu vận hành.
```

# Phương thức

**Phương thức (Method)** là các hàm được định nghĩa bên trong phần thân của một lớp. Chúng được sử dụng để xác định các hành vi của một đối tượng.

### Ví dụ về Class và Method

```
class Car:

    # thuộc tính đối tượng
    def __init__(self, tenxe, mausac, nguyenlieu):
      self.tenxe = tenxe
      self.mausac = mausac
      self.nguyenlieu = nguyenlieu

    # phương thức
    def dungxe(self, mucdich):
      return "{} đang dừng xe để {}".format(self.tenxe,mucdich)

    def chayxe(self):
      return "{} đang chạy trên đường".format(self.tenxe)

 def nomay(self):
      return "{} đang nổ máy".format(self.tenxe)

# instantiate the Car class
toyota = Car("Toyota", "Đỏ", "Điện")
lamborghini = Car("Lamborghini", "Vàng", "Deisel")
porsche = Car("Porsche", "Xanh", "Gas")

# call our instance methods
print(toyota.dungxe("nạp điện"))
```

```
print(lamborghini.chayxe())
print(porsche.nomay())
```

Chạy chương trình, màn hình sẽ trả về kết quả:

```
Toyota đang dừng xe để nạp điện
Lamborghini đang chạy trên đường
Porsche đang nổ máy
```

# Kế thừa (Inheritance)

**Tính kế thừa** cho phép một lớp (class) có thể kế thừa các thuộc tính và phương thức từ các lớp khác đã được định nghĩa. Lớp đã có gọi là lớp cha, lớp mới phát sinh gọi là lớp con. Lớp con kế thừa tất cả thành phần của lớp cha, có thể mở rộng các thành phần kế thừa và bổ sung thêm các thành phần mới.

```
# Lớp cha
class Car:

    # Constructor
    def __init__(self, hangxe, tenxe, mausac):
      # Lớp Car có 3 thuộc tính: tenxe, mausac, hang xe
      self.hangxe = hangxe
      self.tenxe = tenxe
      self.mausac = mausac

    # phương thức
    def chayxe(self):
      print ("{} đang chạy trên đường".format(self.tenxe))

    def dungxe(self, mucdich):
      print ("{} đang dừng xe để {}".format(self.tenxe, mucdich))

# Lớp Toyota mở rộng từ lớp Car.
class Toyota(Car):

    def __init__(self, hangxe, tenxe, mausac, nguyenlieu):
 # Gọi tới constructor của lớp cha (Car)
       # để gán giá trị vào thuộc tính của lớp cha.
      super().__init__(hangxe, tenxe, mausac)

       self.nguyenlieu = nguyenlieu

    # Kế thừa phương thức cũ
    def chayxe(self):
      print ("{} đang chạy trên đường".format(self.tenxe))

    # Ghi đè (override) phương thức cùng tên của lớp cha.
    def dungxe(self, mucdich):
      print ("{} đang dừng xe để {}".format(self.tenxe, mucdich))
      print ("{} chạy bằng {}".format(self.tenxe, self.nguyenlieu))
```

```
 # Bổ sung thêm thành phần mới
    def nomay(self):
      print ("{} đang nổ máy".format(self.tenxe))

toyota1 = Toyota("Toyota", "Toyota Hilux", "Đỏ", "Điện")
toyota2 = Toyota("Toyota", "Toyota Yaris", "Vàng", "Deisel")
toyota3 = Toyota("Toyota", "Toyota Vios", "Xanh", "Gas")

toyota1.dungxe("nạp điện")
toyota2.chayxe()
toyota3.nomay()
```

Kết quả trả về:

```
Toyota Hilux đang dừng xe để nạp điện
Toyota Hilux chạy bằng Điện
Toyota Yaris đang chạy trên đường
Toyota Vios đang nổ máy
```

# Đóng gói (Encapsulation)

Trong Python, chúng ta có thể hạn chế quyền truy cập vào trạng thái bên trong của đối tượng. Điều này ngăn chặn dữ liệu bị sửa đổi trực tiếp, được gọi là **đóng gói**. Trong Python, chúng ta biểu thị thuộc tính private này bằng cách sử dụng dấu gạch dưới làm tiền tố: "_" hoặc "__".

```
class Computer:

    def __init__(self):
      # Thuộc tính private ngăn chặn sửa đổi trực tiếp
      self.__maxprice = 900

   def sell(self):
      print("Giá bán sản phẩm: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
      self.__maxprice = price

c = Computer()c.sell()

# thay đổi giá.
c.__maxprice = 1000
c.sell()

# sử dụng hàm setter để thay đổi giá.
c.setMaxPrice(1000)
c.sell()
```

Màn hình hiển thị kết quả:

```
Selling Price: 900
Selling Price: 900
Selling Price: 1000
```

# Đa hình (Polymorphism)

Tính đa hình là khái niệm mà hai hoặc nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách thức khác nhau.

```
class Toyota:

    def dungxe(self):
      print("Toyota dừng xe để nạp điện")

    def nomay(self):
      print("Toyota nổ máy bằng hộp số tự động")

class Porsche:

    def dungxe(self):
      print("Porsche dừng xe để bơm xăng")

    def nomay(self):
      print("Porsche nổ máy bằng hộp số cơ")

# common interface
def kiemtra_dungxe(car): car.dungxe()

# instantiate objects
toyota = Toyota()
porsche = Porsche()

# passing the object
kiemtra_dungxe(toyota)
kiemtra_dungxe(porsche)
```

Ở ví dụ này, bạn vừa tạo hai lớp *Toyota* và *Porsche*, cả hai lớp đều có phương thức *dungxe().* Truy nhiên hàm của chúng khác nhau. Ta sử dụng tính đa hình để tạo hàm chung cho hai lớp, đó là *kiemtra_dungxe().* Tiếp theo, chúng ta truyền đối tượng *toyota* và *porsche* vào hàm vừa tạo, và ta lấy được kết quả như này:

```
Toyota dừng xe để nạp điện
Porsche dừng xe để bơm xăng
```

# Constructor trong Python

Hàm trong Class được bắt đầu với dấu gạch dưới kép (__) là các hàm đặc biệt, mang các ý nghĩa đặc biệt.

Một trong đó là hàm *__init__()*. Hàm này được gọi bất cứ khi nào khởi tạo một đối tượng, một biến mới trong class và được gọi là constructor trong lập trình hướng đối tượng.

```
class SoPhuc:

    def __init__(self,r = 0,i = 0):
      self.phanthuc = r
      self.phanao = i

    def getData(self):
      print("{}+{}j".format(self.phanthuc,self.phanao))

# Tạo đối tượng số phức mới
c1 = SoPhuc(2,3)

# Gọi hàm getData()
# Output: 2+3j
c1.getData()

# Tạo đối tượng số phức mới
# tạo thêm một thuộc tính mới (new)
c2 = SoPhuc(5)
c2.new = 10

# Output: (5, 0, 10)
print((c2.phanthuc, c2.phanao, c2.new))

# Đối tượng c1 không có thuộc tính 'new'
# AttributeError: 'SoPhuc' object has no attribute 'new'
c1.new
```

Trong ví dụ trên, chúng ta khai báo một lớp mới để biểu diễn các số phức. Nó có hai hàm, *__init__*() để khởi tạo các biến (mặc định là 0) và *getData()* để hiển thị đúng số.

# Kế thừa (Inheritance) trong Python

### Cú pháp của kế thừa

```
class BaseClass:
    Body of base class

class DerivedClass(BaseClass):
    Body of derived class
```

```
class DaGiac:

    def __init__(self, socanh):
      self.n = socanh
      self.canh = [0 for i in range(socanh)]

    def nhapcanh(self):
```

```
        self.canh = [float(input("Bạn hãy nhập giá trị cạnh "+str(i+1)+" : ")) for i in range(self.n)]

    def hienthicanh(self):
        for i in range(self.n):
        print("Giá trị cạnh",i+1,"là",self.canh[i])
```

```
class TamGiac(DaGiac):

    def __init__(self):
        DaGiac.__init__(self,3)

    def dientich(self):
        a, b, c = self.canh
        # Tính nửa chu vi
        s = (a + b + c) / 2
        area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
        print('Diện tích của hình tam giác là %0.2f' %area)
```

Lớp *TamGiac* không chỉ kế thừa mà còn định nghĩa thêm một hàm mới là hàm *dientich.*

```
t = TamGiac()
t.nhapcanh()
Bạn hãy nhập giá trị cạnh 1 : 3
Bạn hãy nhập giá trị cạnh 2 : 5
Bạn hãy nhập giá trị cạnh 3 : 4

t.hienthicanh()
Giá trị cạnh 1 là 3.0
Giá trị cạnh 2 là 5.0
Giá trị cạnh 3 là 4.0

t.dientich()
Diện tích của hình tam giác là 6.00
```

# Overriding (Ghi đè) trong Python

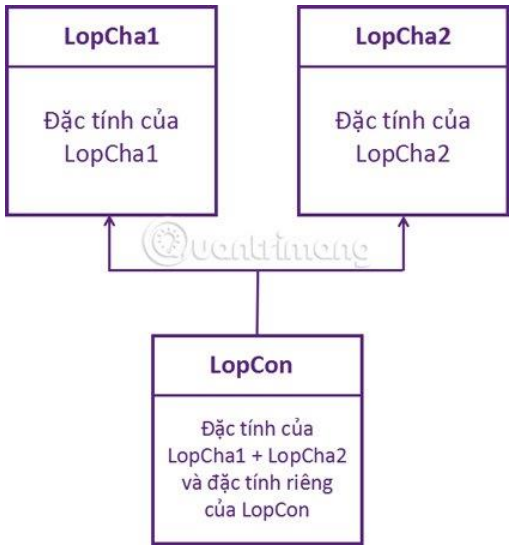
Python cho phép ghi đè lên các phương thức của lớp cha

```
>>> isinstance(t,TamGiac)
True

>>> isinstance(t,DaGiac)
True

>>> isinstance(t,int)
False

>>> isinstance(t,object)
True
```

# Đa kế thừa (Multiple Inheritance) trong Python

**Cú pháp:**

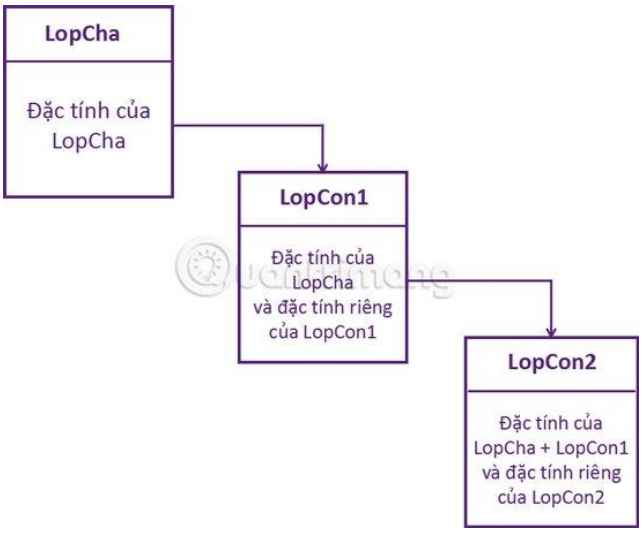
```
class LopCha1:
    Khai báo lớp

class LopCha2:
    Khai báo lớp

class LopCon(LopCha1, LopCha2):
    Khai báo lớp
```




# Kế thừa đa cấp (Multilevel Inheritance)

**Cú pháp:**

```
class LopCha:
    pass

class LopCon1(LopCha):
    pass

class LopCon2(LopCon1):
    pass
```




# Nạp chồng toán tử trong Python

## Ví dụ: Nạp chồng toán tử ' + ' trong Python

Để nạp chồng toán tử ' + ', ta sẽ sử dụng hàm *__add__()* trong class. Ta có thể triển khai nhiều công việc bằng hàm này, ví dụ như cộng hai điểm tọa độ ở ví dụ bên trên.

```
class Point:
    def __init__(self, x = 0, y = 0):
      self.x = x
      self.y = y

    def __str__(self):
      return "({0},{1})".format(self.x,self.y)

    def __add__(self,other):
      x = self.x + other.x
      y = self.y + other.y
      return Point(x,y)
```

```
>>> p1 = Point(2,3)
>>> p2 = Point(-1,2)
>>> print(p1 + p2)
(1,5)
```

Một số hàm đặc biệt dùng cho nạp chồng toán tử trong bảng dưới đây:

| TOÁN TỬ | BIỂU DIỄN | HOẠT ĐỘNG |
|---|---|---|
| Phép cộng | p1 + p2 | p1.add(p2) |
| Phép trừ | p1 - p2 | p1.sub(p2) |
| Phép nhân | p1 * p2 | p1.mul(p2) |
| Lũy thừa | p1 ** p2 | p1.__pow__(p2) |
| Phép chia | p1 / p2 | p1.truediv(p2) |
| Phép chia lấy phần nguyên (Floor Division) | p1 // p2 | p1.__floordiv__(p2) |
| Số dư (modulo) | p1 % p2 | p1.mod(p2) |
| Thao tác trên bit: phép dịch trái | p1 << p2 | p1.lshift(p2) |
| Thao tác trên bit: phép dịch phải | p1 >> p2 | p1.__rshift__(p2) |
| Thao tác trên bit: phép AND | p1 & p2 | p1.and(p2) |
| Thao tác trên bit: phép OR | p1 \| p2 | p1.or(p2) |
| Thao tác trên bit: phép XOR | p1 ^ p2 | p1.xor(p2) |
| Thao tác trên bit: phép NOT | ~p1 | p1.__invert__() |

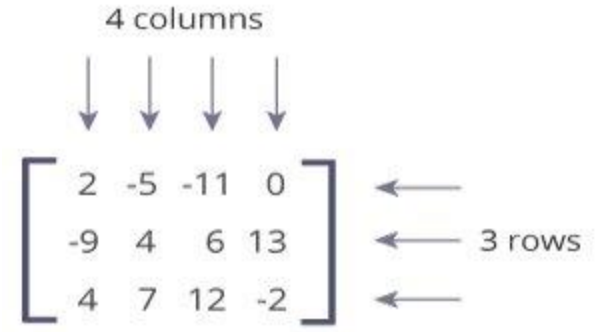
# Nạp chồng toán tử so sánh trong Python

Có nhiều toán tử so sánh được hỗ trợ bởi Python, ví dụ như: <, >, <=, >=, ==,...

```
class Point:
    def __init__(self, x = 0, y = 0):
      self.x = x
      self.y = y

    def __str__(self):
      return "({0},{1})".format(self.x,self.y)

    def __lt__(self,other):
      self_mag = (self.x ** 2) + (self.y ** 2)
      other_mag = (other.x ** 2) + (other.y ** 2)
      return self_mag < other_mag
```

```
>>> Point(1,1) < Point(-2,-3)
True

>>> Point(1,1) < Point(0.5,-0.2)
False

>>> Point(1,1) < Point(1,1)
False
```

# 9. Ma trận trong Python




Đây là ma trận 3x4 vì nó có 3 hàng và 4 cột.

Python không có kiểu xây dựng riêng dành cho ma trận, vậy nên chúng ta có thể biểu diễn ma trận dưới dạng một nested list.

**Nested list** là dạng danh sách lồng ghép, nghĩa là một list xuất hiện với vai trò là phần tử của một list khác. Ví dụ:

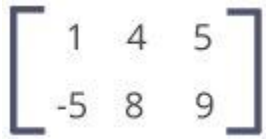
```
A = [ 1, 4, 5, [8, 9]]
```

Ở ví dụ này, nếu in ra A[3] ta được output là [8, 9].

Nested list thường được dùng để trình bày ma trận trong Python. Biểu diễn như sau:

```
A = [[1, 4, 5],
   [-5, 8, 9]]
```

Chúng ta có thể coi danh sách này là một ma trận gồm 2 hàng và 3 cột.




Ví dụ

```
A = [[1, 4, 5, 12],
 [-5, 8, 9, 0],
 [-6, 7, 11, 19]]

print("A =", A)
print("A[1] =", A[1]) # Hàng thứ 2 của ma trân
print("A[1][2] =", A[1][2]) # Phần tử thứ 3 của hàng thứ 2
print("A[0][-1] =", A[0][-1]) # Phần tử cuối cùng của hàng 1

column = [];
for row in A:
 column.append(row[2])
```

```
print("Cột thứ 3 =", column)
```

Chạy chương trình, output được trả về là:

```
A = [[1, 4, 5, 12], [-5, 8, 9, 0], [-6, 7, 11, 19]]
A[1] = [-5, 8, 9, 0]
A[1][2] = 9
A[0][-1] = 12
Cột thứ 3 = [5, 9, 11]
```

# Sử dụng NumPy cho ma trận

**NumPy** là thư viện được viết bằng Python nhằm phục vụ cho việc tính toán khoa học, hỗ trợ nhiều kiểu dữ liệu đa chiều giúp cho việc tính toán, lập trình, làm việc với các hệ cơ sở dữ liệu cực kì thuận tiện.

Để tạo một ma trận ta có thể sử dụng ndarray (viết gọn là array) của NumPy

Array này là một đối tượng mảng đa chiều thuần nhất tức là mọi phần tử đều cùng 1 kiểu.

```
import numpy as np
a = np.array([1, 2, 3])
print(a)

# Output: [1, 2, 3]

print(type(a))
```

# Cách tạo array của NumPy

```
import numpy as np

A = np.array([[1, 2, 3], [3, 4, 5]])
print(A)

A = np.array([[1.1, 2, 3], [3, 4, 5]]) # mảng số thực
print(A)

A = np.array([[1, 2, 3], [3, 4, 5]], dtype = complex) # mảng số phức
print(A)
```

Chương trình trả về kết quả:

```
[[1 2 3]
 [3 4 5]]
```

```
[[1.1 2. 3. ]
 [3. 4. 5. ]]

[[1.+0.j 2.+0.j 3.+0.j]
 [3.+0.j 4.+0.j 5.+0.j]]
```

**Mảng giá trị mặc định (0 và 1)**

```
import numpy as np

# Mọi phần tử đều là 0
A = np.zeros( (2, 3) )
print(A)

# Output:
 [[0. 0. 0.]
 [0. 0. 0.]]

# Mọi phần tử đều là 1
B = np.ones( (1, 5) )
print(B)

# Output: [[1 1 1 1 1]]
```

# Các phép toán với ma trận

**Cộng 2 ma trận**

Để cộng hai ma trận, ta cộng từng phần tử tương ứng của 2 ma trận cùng cấp với nhau

```
import numpy as np

A = np.array([[2, 4], [5, -6]])
B = np.array([[9, -3], [3, 6]])
C = A + B
print(C)

'''
Output:
[[11 1]
 [ 8 0]]
 '''
```

**Nhân 2 ma trận**

```
import numpy as np

A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = a.dot(B)
print(C)
```

```
# Output:
[[ 36 -12]
 [ -1 2]]
```

### Chuyển vị ma trận

```
import numpy as np

A = np.array([[1, 1], [2, 1], [3, -3]])
print(A.transpose())

#Output:
[[ 1 2 3]
 [ 1 1 -3]]
```

### Xuất các phần tử của ma trận

```
import numpy as np
A = np.array([12, 14, 16, 18, 20])

print("A[0] =", A[0]) # phần tử đầu tiên
print("A[2] =", A[2]) # phần tử thứ 3
print("A[-1] =", A[-1]) # phần tử cuối cùng
```

Output được trả về ở đây là:

```
A[0] = 12
A[2] = 16
A[-1] = 20
```

Ví dụ về mảng hai chiều:

```
import numpy as np

A = np.array([[1, 4, 5, 12],
 [-5, 8, 9, 0],
 [-6, 7, 11, 19]])

# Phần tử đầu tiên của hàng đầu tiên
print("A[0][0] =", A[0][0])

# Phần tử thứ 3 của hàng thứ 2
print("A[1][2] =", A[1][2])

# Phần tử cuối cùng của hàng cuối cùng
print("A[-1][-1] =", A[-1][-1])
```

Chạy chương trình, kết quả được trả về là:

```
A[0][0] = 1
A[1][2] = 9
A[-1][-1] = 19
```

### Xuất các dòng của ma trận

```
import numpy as np

A = np.array([[1, 4, 5, 12],
 [-2, 8, 6, 14],
 [-1, 5, 10, 22]])

print("A[0] =", A[0]) # Dòng đầu tiên
print("A[2] =", A[2]) # Dòng thứ 3
print("A[-1] =", A[-1]) # Dòng cuối cùng (dòng thứ 3)
```

Output được trả về ở đây là:

```
A[0] = [1, 4, 5, 12]
A[2] = [-1, 5, 10, 22]
A[-1] = [-1, 5, 10, 22]
```

### Xuất các cột của ma trận

```
import numpy as np

A = np.array([[1, 4, 5, 12],
 [-2, 8, 6, 14],
 [-1, 5, 10, 22]])

print("A[:,0] =",A[:,0]) # Cột đầu tiên
print("A[:,3] =", A[:,3]) # Cột thứ 4
print("A[:,-1] =", A[:,-1]) # Cột cuối cùng (Cột thứ 4)
```

Output được trả về:

```
A[:,0] = [ 1 -2 -1]
A[:,3] = [12 14 22]
A[:,-1] = [12 14 22]
```

# BÀI TẬP

**1.** Viết hàm trong chương trình Python in ra dãy số Fibonacci giữa 0 và 50 (sử dụng hàm đệ quy)

**2.** Viết một hàm Python tính tổng tất cả các số trong một danh sách.

**3.** Viết một hàm Python đầu vào là một số và kiểm tra xem số đó có phải là số nguyên tố hay không.

4. Khởi tạo 2 ma trận 3x3, áp dụng các phép toán cộng, trừ 2 ma trận và xuất ra ma trận kết quả.

**5. Xây dựng lớp KhachHang**

Khách hàng của ngân hàng XYZ có một số thông tin: mã khách hàng, họ tên khách hàng, năm sinh, địa chỉ, email, điện thoại, tiền trong tài khoản

- Khai báo các fields
- Thêm các constructors :
  - Tạo constructor không tham số (default constructor)
- Thêm phương thức NhapThongTin() để nhập thông tin khách hàng từ bàn phím
- Thêm phương thức XuatThongTin() để xuất thông tin khách hàng ra màn hình
- Thêm phương thức TinhTuoi() để tính tuổi của khách hàng so với năm hiện tại trên máy tính.
- Sau đó
  - Tạo đối tượng khách hàng và nhập thông tin từ bàn phím
  - Xuất thông tin khách hàng có kèm theo tuổi của khách hàng

```
Nhap thong tin khách hàng
- Ma khach hang: NV1234
- Ho ten: Nguyen Van Lam
- Nam sinh: 1990
- Dia chi: 25 Nguyen Trai, P5, Q1, TP.HCM
- Email: nguyenvanlam@gmail.com
- Dien thoai: 090123456
- So tien trong tai khoan: 4500000

In thong tin khach hang
- Ma khach hang: NV1234
- Ho ten: Nguyen Van Lam
- Nam sinh: 1990
- Dia chi: 25 Nguyen Trai, P5, Q1, TP.HCM
- Email: nguyenvanlam@gmail.com
- Dien thoai: 090123456
- So tien trong tai khoan: 4500000
- Tuoi: 29
```

**6. Kế thừa – Đa hình:** Phân tích, thiết kế và hiện thực theo hướng đối tượng chương trình để thông báo các thành viên được khen thưởng trong năm học của một trường đại học. Thành viên trong trường đại học gồm: **Giảng viên, Sinh viên.** Thông tin cơ bản gồm: mã (mã sinh viên/ mã giảng viên), họ tên và bộ phận trực thuộc (Khoa nào, phòng ban nào).

Tiêu chuẩn xét khen thưởng:

- **Sinh viên:** Điểm trung bình tích lũy phải >=8
- **Giảng viên:** có ít nhất hai công trình nghiên cứu khoa học

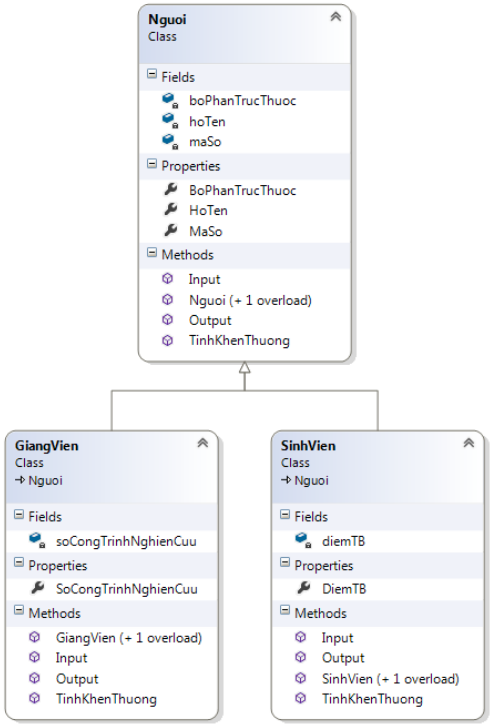
***Yêu cầu:***

a) Hãy thiết kế các lớp cho bài toán sao cho phương thức nhập xuất thông tin **Input(), Output()** và phương thức khen thưởng **TinhKhenThuong()** có tính đa hình

b) Hãy cài đặt sơ đồ lớp trên sao cho các phương thức **Input(), Output() và TinhKhenThuong()** có tính đa hình

c) Tạo 1 đối tượng **Sinh viên** và 1 đối tượng **Giảng viên**

d) Nhập và Xuất thông tin.




## **Bài 7.** Xây dựng lớp **Tam giác và** lớp **Chữ nhật**

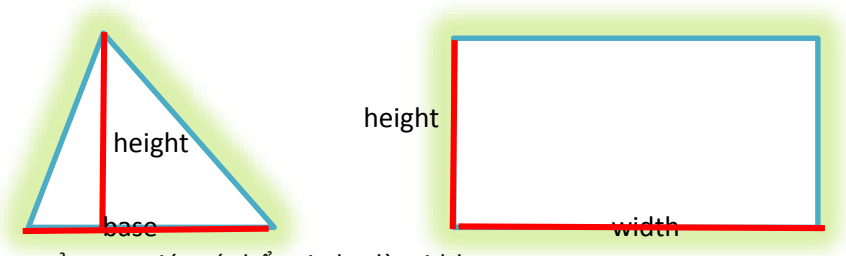
**a. Lớp cơ sở**: Lớp hình (**Shape**)

Xây dựng lớp hình (**Shape**) dùng để mô hình hóa một hình gồm có thông tin chiều cao (**height**) và chiều rộng (**width**) của hình.

- Khai báo các **Fields**
- Thêm các constructors :
  - Tạo constructor không tham số (default constructor)

**b. Lớp dẫn xuất**: Xây dựng lớp tam giác (**Triangle)** và lớp hình chữ nhật (**Rectangle)**

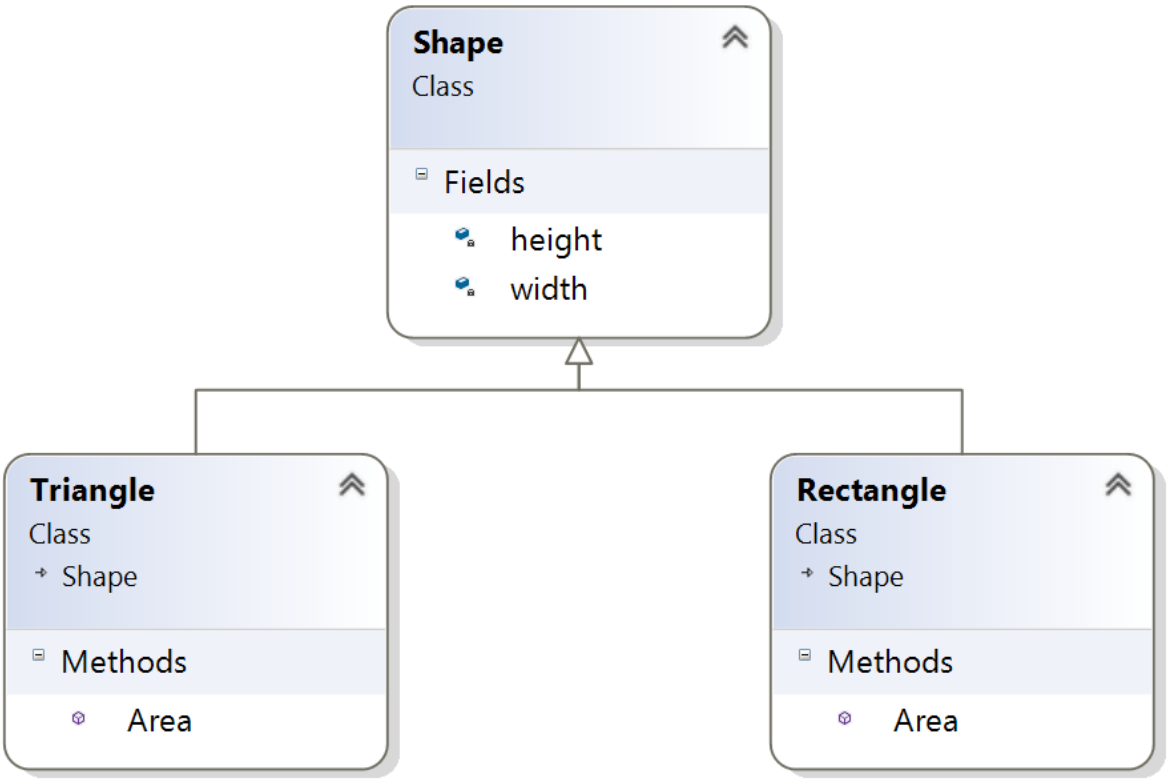
Hình Tam giác và hình chữ nhật cũng là một hình có các thông tin như sau:




Trong đó tham số base của tam giác có thể coi như là width

Xây dựng 2 lớp dẫn xuất là lớp hình tam giác (**Triangle**) và lớp hình chữ nhật (**Rectangle**) **thừa kế từ** lớp **Shape**.

Sơ đồ thừa kế




- Cài đặt **default constructor**
- Viết phương thức nhập dữ liệu **Input()** cho lớp hình chữ nhật và hình tam giác sao cho phương thức này có tính đa hình
- Viết phương thức xuất dữ liệu **Output()** cho lớp hình chữ nhật và hình tam giác sao cho phương thức này có tính đa hình

- Cài đặt thêm phương thức **Area()** có tính đa hình để tính diện tích các hình tương ứng

$$S_{tam\ giác} = \frac{1}{2} \times height \times width$$

$$S_{chữ\ nhật} = height \times width$$

- **Nhập xuất thông tin**
  - Tạo một danh sách chứa một số đối tượng **Rectangle**, **Triangle** (dùng phương thức constructor để tạo đối tượng)
  - Nhập n hình khác nhau (n nhập từ bàn phím).
  - Xuất thông tin các hình đã nhập
  - Tính và xuất diện tích từng hình đã nhập. Tính và xuất diện tổng diện tích các hình đã nhập

Phương thức **Main()** hoạt động cho kết quả có dạng như sau:

```
Nhap so hinh n: 2
Nhap hinh thu 1:
Loai hinh (Neu hinh chu nhat nhap 'C', hinh tam giac nhap 'T'): C
Height: 3
Width: 4

Nhap hinh thu 2:
Loai hinh (Neu hinh chu nhat nhap 'C', hinh tam giac nhap 'T'): T
Base: 7
Height: 9

Cac hinh da nhap:
Hinh chu nhat: Height=3, Width=4
Hình tam gia: Height=9, Base = 7

Dien tich hinh chu nhat: …
Dien tich hinh tam giac: …
Tong dien tich: …
```