# BÀI TẬP

## Lớp và OOP

*Mục tiêu: Làm quen với lớp (class), thiết kế các lớp theo hướng đối tượng.*

# Bài tập thực hành

### Bài 1:

Tạo lớp StringExp, lớp này có một thuộc tính là exp, và một phương thức là isValid, trả lại thông tin về tính hợp lệ về dấu ngoặc của chuỗi exp. Chuỗi exp được gọi là hợp lệ về dấu ngoặc '(', ')', '{', '}', '[' và '] nếu các dấu ngoặc mở và đóng đúng thứ tự, ví dụ "()" and "()[]{}" là hợp lệ nhưng "[)", "({[)]" and "{{{" là không hợp lệ.

### Bài 2

Viết lớp WellNetExp, lớp này kế thừa từ lớp StringExp, và có thêm 1 phương thức isWellNet, phương thức này trả về thông tin về tính đầy đủ về dấu ngoặc '(' ')' của biểu thức exp. Biểu thức exp được gọi là đầy đủ dấu ngoặc nếu nó hợp lệ về dấu ngoặc và mọi phép toán trong biểu thức đều được đặt trong một cặp ngoặc '()'. Ví dụ: biểu thức: (((1+2)+3)+4) là biểu thức đầy đủ dấu ngoặc, biểu thức sau không đầy đủ dấu ngoặc: (1+2+3+4). Lớp WellNetExp có thêm một phương thức nữa là compute, phương thức này thực hiện tính và trả lại giá trị của biểu thức đầy đủ dấu ngoặc.

### Bài 3:

Viết lớp NormalExp, lớp này kế thừa từ lớp StringExp, lớp có phương thức compute để tính giá trị biểu thức exp, nếu biểu thức exp là hợp lệ về dấu ngoặc.

Thuật toán để tính giá trị biểu thức thông qua biểu thức dạng hậu tố:

Đầu tiên ta cần chuyển biểu thức từ dạng trung tố sang hậu tố, thuật toán như sau: ta xét lần lượt từng phần tử (token) trên biểu thức từ trái qua phải, tại mỗi phần tử:

*Nếu là toán hạng: cho ra output.*

– *Nếu là dấu mở ngoặc "(": cho vào stack*

– *Nếu là dấu đóng ngoặc ")": lấy các toán tử trong stack ra và cho vào output cho đến khi gặp dấu mở ngoặc "(". (Dấu mở ngoặc cũng phải được đưa ra khỏi stack)*

– *Nếu là toán tử:*

- *Chừng nào ở đỉnh stack là toán tử và toán tử đó có độ ưu tiên* ***lớn hơn hoặc bằng*** *toán tử hiện tại thì lấy toán tử đó ra khỏi stack và cho ra output.*
- *Đưa toán tử hiện tại vào stack*

*Sau khi duyệt hết biểu thức infix, nếu trong stack còn phần tử thì lấy các token trong đó ra và cho lần lượt vào output.*

Tiếp đến ta cần tính giá trị của biểu thức dạng hậu tố theo giải thuật sau:

*Lặp qua các token của của biểu thức postfix từ trái qua phải:*

– *Nếu là toán hạng: push vào stack*

– *Nếu là toán tử: pop hai toán hạng trong stack ra và tính giá trị của chúng dựa vào toán tử này. Push kết quả đó lại vào stack.*

*Phần tử còn sót lại trong stack sau vòng lặp chính là kết quả của biểu thức.*

## Bài 4:

Một công ty phần mềm có nhiều nhân viên (Employee), mỗi tháng công ty cần tính và trả lương cho nhân viên của mình. Tuy nhiên, công việc tính lương không hề đơn giản, nó đòi hỏi rất nhiều thời gian và tính toán cẩn thận. Bạn hãy xây dượng một chương trình giúp công ty giải quyết vấn đề trên. Biết rằng mỗi nhân viên có một mã số, họ tên, năm sinh và một mức lương cơ bản (MLCB). Công thức tính lương như sau:

- Nếu là quản lý (Manager) thì tiền lương sẽ được tính bằng MLCB + MLCB x 25%
- Nếu là nhà phân tích dữ liệu (Data Scientist) thì tiền lương sẽ được tính bằng MLCB + MLCBx20% + số dự án tham gia trong tháng x 1500$
- Nếu là lập trình viên (Developer) thì tiền lương sẽ được tính bằng MLCB + số dự án thực hiện trong tháng x 1000$
- Các nhân viên khác được trả lương bằng MLCB

Xây dựng các lớp hợp lý, tạo lớp EmployeeManager để quản lý thông tin các nhân viên trong công ty.

## Bài 5:

Viết lớp tổng quát có thể nhận mọi thuộc tính và giá trị thuộc tính từ người dùng. Trong lớp có hàm __str__ thực hiện in ra các thuộc tính của đối tượng cũng như giá trị của các thuộc tính đó.

Code mẫu bài 5:

```
class AnyClass:
    def __init__(self, **kwargs):
        for k, v in kwargs.items():
            setattr(self, k, v)

    def __str__(self):
        attrs = ["%s=%s" % (k, v) for (k, v) in self.__dict__.items()]
        classname = self.__class__.__name__
        return "%s: %s" % (classname, " ".join(attrs))

sample = AnyClass(name = "Nam", age = 18)
print(sample)
sample1 = AnyClass(Brand = "HONDA", name = "SH", price = 10000)
print(sample1)
```

Trong ví dụ trên ta sử dụng đối số với 2 dấu ** để khai báo là hàm __ `init` __ sẽ không giới hạn về số đối, mỗi đối số là 1 cặp Khóa = Giá trị.

Bạn hãy viết thêm 1 hàm anyParas, hàm này in ra các giá trị của đối số *pars. Ta dùng đối số với 1 dấu * để khai báo rằng hàm này không giới hạn về số đối và mỗi đối số là 1 giá trị.

## Bài 6:

Viết lớp tên là `Numbers`, lớp này có một thuộc tính lớp là `MULTIPLIER`, và hàm dựng nhận hai đối số là `x` và `y` (các đối này là các số).

1. Viết phương thức `add` trả lại tổng của `x` và `y`.
2. Viết phương thức `multiply`, phương thức này nhận vào đối số là `a` và trả lại tích của `a` và `MULTIPLIER`.
3. Viết phương thức tĩnh `subtract`, phương thức này nhận vào hai đối là, `b` và `c`, và kết quả trả lại là `b - c`.
4. Viết phương thức `value` phương thức này trả lại bộ giá trị ( `x` và `y)`. Chuyển phuong thức này thành **property**, viết setter và để điều khiển giá trị của `x` và `y`.

Ví dụ mẫu bài 6

```
class Numbers:
    MULTIPLIER = 3.5

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def add(self):
        return self.x + self.y

    @classmethod
    def multiply(cls, a):
        return cls.MULTIPLIER * a

    @staticmethod
    def subtract(b, c):
        return b - c

    @property
    def value(self):
        return (self.x, self.y)

    @value.setter
    def value(self, xy_tuple):
        self.x, self.y = xy_tuple

    @value.deleter
    def value(self):
        del self.x
        del self.y

a = Numbers(1,2)
print(a.value)
a.value = 3,4
print(a.value)

print(a.value)

print(Numbers.subtract(1, 2))

del a.value
```