```
!date
```

```
Tue Feb  8 06:21:45 UTC 2022
```

Topic 1 (Creating a class)

```
class Student:
    # Constructor / Initializer
    # name should be stored publicly;
    # age should be stored publicly;
    # GPA should be stored privately. (By convention)
    def __init__(self, name, age, GPA):
        #solution
        self.name = name
        self.age = age
        self._GPA = GPA

    # For private variables we need getters/setters. By convention.
    def get_GPA(self):
        #solution
        return self._GPA


    def set_GPA(self, GPA):
        #solution
        self._GPA = GPA


def main():
    bob = Student("Bob", 15, 3.0)
    print(bob.get_GPA()) #3.0

    bob.set_GPA(4.0)
    print(bob.get_GPA()) #4.0

if __name__ == '__main__':
    main()
```

```
3.0
4.0
```

What does the keyword self do in Python?

Answer: self represents the current instance of the class

Topic 2 (underscore ***** functions):

```
class  Pizza:
       def  __init__(self,  price):
              self.price  =  price

       def  __add__(self,  other):
              new_pizza  =  Pizza(self.price)
              new_pizza  +=  other
              return  new_pizza

       def  __iadd__(self,  other):
              self.price  +=  other.price
              return  self

       def  __str__(self):
              return  "the  price  is,  "  +  str(self.price)



def  main():
       pizza1  =  Pizza(5)
       pizza2  =  Pizza(6)
       pizza1  +  pizza2
       pizza1  +=  pizza2
       print(pizza1)

if  __name__  ==  '__main__':
       main()
```

```
the price is, 11
```

a) What does the code above print? Don't run the program, try to predict the output first.

Answer: the price is, 11

b) Complete the following table, suppose the variable name is X. When will these underscore functions get called? Answer for 1st row has been given for your convenience.

Answer:

1. X.**getitem**(self, index) : X[index]
2. X.**setitem**(self, index, value) : X[index] = value
3. X.**delitem**(self, index) : del X[index]
4. X.**add**(self, other) : X + other
5. X.**iadd**(self, other) : X += other
6. X.**eq**(self, other) : X == other
7. X.**len**(self) : len(X)

8. X.**str**(self) : print(X)
9. X.**repr**(self) : print(X)
10. X.**contains**(self, value) : value in X
11. X.**iter**(self) : for each in X:

Topic 3 (Inheritance):

```
class Tree:
    def __init__(self, name, age):
        self._name = name
        self._age = age

    def get_name(self):
        return self._name


class Palm(Tree): # Palm(Tree) means, Palm inherits Tree.
    def __init__(self, name, age, color):
        # First you have to initialize the parent class. What should we write
        # Solution
        super().__init__(name, age)

        self._color = color

    def get_color(self):
        return self._color

def main():
    palm1 = Palm("Lucky", 30, "Green")
    print(palm1.get_name())   # What does this print (1)?
    print(palm1.get_color())   # What does this print (2)?
    tree1 = Tree("Funny", 20)
    print(tree1.get_name())   # What does this print (3)?
    print(tree1.get_color())   # What does this print (4)?

if __name__ == '__main__':
    main()
```

```
Lucky
Green
Funny
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-8-2c199fc11e4f> in <module>()
     28
     29 if __name__ == '__main__':
```

What does the code above print? Don't run the program, try to predict the output first.

```
<ipython-input-8-2c199fc11e4f> in main()
```

1. What is the output for print (1)? **Answer:** lucky
2. What is the output for print (2)? **Answer:** Green
3. What is the output for print (3)? **Answer:** Funny
4. What is the output for print (4)? **Answer:** Error

Topic 4 (Misc):

```
# Coding 1
# Creating a class using UML Diagram, and then use this class.
# Now implement the following class hierarchy: Shape, Circle, and Rectangle.
# Note: Both Circle, Rectangle inherits Shape. Once finished, test your Shape, Circle,

class Shape:
    def __init__(self, name):
        self.name = name

    def get_name(self):
        # Solution
        return self.name

class Circle(Shape):
    def __init__(self, name, radius):
        # Solution
        super().__init__(name)
        self.radius = radius

    def calc_area(self):
        # Solution
        return self.radius ** 2 * 3.14

    def calc_perimeter(self):
        # Solution
        return self.radius * 2 * 3.14

class Rectangle(Shape):
    def __init__(self, name, width, height):
        # Solution
        super().__init__(name)
```

```
                self.width = width
                self.height = height

        def calc_area(self):
                # Solution
                return self.width * self.height

        def calc_perimeter(self):
                # Solution
                return 2 * (self.width + self.height)

def main():
        circle1 = Circle("fancy", 5)
        print(circle1.calc_area())    #78.5
        print(circle1.calc_perimeter())  #31.400000000000002

        rectangle1 = Rectangle("lucky", 3, 4)
        print(rectangle1.calc_area())    #12
        print(rectangle1.calc_perimeter())    #14

if __name__ == '__main__':
        main()
```

```
78.5
31.400000000000002
12
14
```

```
# Coding 2
# Design and implement class Polynomial. It should behave like the following way:
# Suppose I want to represent 1x^4 + 2x^3 + 3x^2 + 4x + 5, then I input python l
# [1,2,3,4,5] into class Polynomial, my Polynomial should represent 1x^4 + 2x^3 + 3x^2
# If I want to evaluate this polynomial, given x = 2, I simply call Polynomial.evalu
# If I want to display my polynomial, I simply print it.
# I should also be able to add one polynomial to another using +=.

class Polynomial:
        def __init__(self, coeffs):
                self.coeffs = coeffs

        # Solution
        def evaluate_at(self, x):
                return sum(self.coeffs[-power - 1]*x**power for power in range(len(self.coef


        #def evaluate_at(self, x):
                #output = 0
                #for power in range(len(self.coeffs)):
                        #output += self.coeffs[-power - 1]*x**power
                #return output

        def __iadd__(self, otherPoly):
                if len(self.coeffs) > len(otherPoly.coeffs):
                        longer, shorter = self.coeffs.copy(), otherPoly.coeffs.copy()
                else:
```

```
                longer, shorter = otherPoly.coeffs.copy(), self.coeffs.copy()
            for i in range(len(shorter)):
                longer[-i - 1] += shorter[-i - 1]
            self.coeffs = longer
            return self

        def __str__(self):
            l = [str(self.coeffs[x]) + "x^" + str(len(self.coeffs) - x - 1) + " +
                 for x in range(len(self.coeffs) - 1)]
            return "".join(l) + str(self.coeffs[-1])


        #def __str__(self):
            #l = []
            #for x in range(len(self.coeffs) - 1):
                #l.append( str(self.coeffs[x]) + "x^" + str(len(self.coeffs) - x -

            #return "".join(l) + str(self.coeffs[-1])


def main():
    # 1x^4 + 2x^3 + 3x^2 + 4x + 5
    coeffs = [1,2,3,4,5]
    poly = Polynomial(coeffs)
    print(poly.evaluate_at(2))      # 57
    print(poly.evaluate_at(3))      # 179
    print(poly)    # Outputs: 1x^4 + 2x^3 + 3x^2 + 4x^1 + 5

    # 4x^3 + 6x^2 + 8x^1 + 10
    coeffs = [4,6,8,10]
    poly2 = Polynomial(coeffs)
    print(poly2)    # Outputs: 4x^3 + 6x^2 + 8x^1 + 10
    poly += poly2
    print(poly)    # Outputs: 1x^4 + 6x^3 + 9x^2 + 12x^1 + 15

if __name__ == '__main__':
    main()
```

```
57
179
1x^4 + 2x^3 + 3x^2 + 4x^1 + 5
4x^3 + 6x^2 + 8x^1 + 10
1x^4 + 6x^3 + 9x^2 + 12x^1 + 15
```

```
# 此内容为代码格式
```

Topic 5 Problem 1 Reverse Digit

```
#Given a 32-bit signed integer, return the reversed digits of this integer.
#Note:
```

```
#Try to solve this problem using math equations.
#Eg: don't cast this number to str/list/etc.


def reverse(x):
        # Solution
        flag = False
        if x < 0:
                flag = True
                x = 0 - x
        result = 0
        while (x > 0):
                result = result * 10 + (x % 10)
                x = x // 10

        if (flag == True):
                result = 0 - result

        return result

# test case
print(reverse(1200)) #21
print(reverse(123)) #321
print(reverse(-123)) #-321
```

```
21
321
-321
```

Problem 2

```
#Write a program to check whether a given number is a Funny number.
#Funny numbers are positive numbers whose prime factors only include 2, 3, 5.
#For example, 6, 8 are Funny while 14 is not Funny since it includes another prime


def isFunny(num):
        # Solution
        if num <= 0:
                return False
        if num == 1:
                return True

        while num >= 2 and num % 2 == 0:
                num /= 2;
        while num >= 3 and num % 3 == 0:
                num /= 3;
        while num >= 5 and num % 5 == 0:
                num /= 5;

        return num == 1
```

```
#  test  case
print(isFunny(6))  #True
print(isFunny(8))  #True
print(isFunny(14))  #False
print(isFunny(1))  #True
```

```
True
True
False
True
```

✓ 0 秒 完成时间：14:22