OOP and Orders of Growth

Computer Science Mentors CS 61A

March 11 to March 13, 2019

1 Object Oriented Programming

1. **(H)OOP**

   Given the following code, what will Python output for the following prompts?

```
class Baller:
    all_players = []
    def __init__(self, name, has_ball = False):
        self.name = name
        self.has_ball = has_ball
        Baller.all_players.append(self)

    def pass_ball(self, other_player):
        if self.has_ball:
            self.has_ball = False
            other_player.has_ball = True
            return True
        else:
            return False

class BallHog(Baller):
    def pass_ball(self, other_player):
        return False
```

```
>>> richard = Baller('Richard', True)
>>> albert = BallHog('Albert')
>>> len(Baller.all_players)

>>> Baller.name

>>> len(albert.all_players)


>>> richard.pass_ball()

>>> richard.pass_ball(albert)

>>> richard.pass_ball(albert)

>>> BallHog.pass_ball(albert, richard)

>>> albert.pass_ball(richard)

>>> albert.pass_ball(albert, richard)
```

2. Write TeamBaller, a subclass of Baller. An instance of TeamBaller cheers on the team every time it passes a ball.

```
class TeamBaller(_______________):
    """
    >>> samy = BallHog('Samy')
    >>> cheerballer = TeamBaller('Mary', has_ball=True)
    >>> cheerballer.pass_ball(samy)
    Yay!
    True
    >>> cheerballer.pass_ball(samy)
    I don't have the ball
    False
    """
    def pass_ball(_______________, ________________):
```

3. Lets use OOP to help us implement our good friend, the ping-pong sequence!

   As a reminder, the ping-pong sequence counts up starting from 1 and is always either counting up or counting down.

   At element k, the direction switches if k is a multiple of 7 or contains the digit 7.

   The first 30 elements of the ping-pong sequence are listed below, with direction swaps marked using brackets at the 7th, 14th, 17th, 21st, 27th, and 28th elements:

   ```
   1 2 3 4 5 6 [7] 6 5 4 3 2 1 [0] 1 2 [3] 2 1 0 [-1] 0 1 2 3 4
   [5] [4] 5 6
   ```

   Assume you have a function `has_seven(k)` that returns True if k contains the digit 7.

```
>>> tracker1 = PingPongTracker()
>>> tracker2 = PingPongTracker()
>>> tracker1.next()
1
>>> tracker1.next()
2
>>> tracker2.next()
1

class PingPongTracker:
    def __init__(self):







    def next(self):
```

4. **Flying the cOOP** What would Python display?
   Write the result of executing the code and the prompts below.
   If a function is returned, write ”Function”.
   If nothing is
   returned, write ”Nothing”. If an error occurs, write ”Error”.

```
class Bird:
    def __init__(self, call):
        self.call = call
        self.can_fly = True
    def fly(self):
        if self.can_fly:
            return "Don't stop
               me now!"
        else:
            return "Ground
               control to Major
               Tom..."
    def speak(self):
        print(self.call)

class Chicken(Bird):
    def speak(self, other):
        Bird.speak(self)
        other.speak()

class Penguin(Bird):
    can_fly = False
    def speak(self):
        call = "Ice to meet you
           "
        print(call)

andre = Chicken("cluck")
gunter = Penguin("noot")
```

```
>>> andre.speak(Bird("coo"))
```

```
>>> andre.speak()
```

```
>>> gunter.fly()
```

```
>>> andre.speak(gunter)
```

```
>>> Bird.speak(gunter)
```

5. What would Python display? Write the result of executing the code and the prompts below. If a function is returned, write ”Function”. If nothing is returned, write ”Nothing”. If an error occurs, write ”Error”.

```
class Musician:
    popularity = 0
    def __init__(self, instrument):
        self.instrument = instrument
    def perform(self):
        print("a rousing " + self.instrument + " performance")
        self.popularity = self.popularity + 2
    def __repr__(self):
        return self.instrument

class BandLeader(Musician):
    def __init__(self):
        self.band = []
    def recruit(self, musician):
        self.band.append(musician)
    def perform(self, song):
        for m in self.band:
            m.perform()
        Musician.popularity += 1
        print(song)
    def __str__(self):
        return "Here's the band!"
    def __repr__(self):
        band = ""
        for m in self.band:
            band += str(m) + " "
        return band[:-1]

miles = Musician("trumpet")
goodman = Musician("clarinet")
ellington = BandLeader()
```

```
>>> ellington.recruit(goodman)
>>> ellington.perform()


>>> ellington.perform("sing, sing, sing")


>>> goodman.popularity, miles.popularity


>>> ellington.recruit(miles)
>>> ellington.perform("caravan")


>>> ellington.popularity, goodman.popularity, miles.popularity


>>> print(ellington)


>>> ellington
```

2 Orders of Growth

1. What is the order of growth in time for the following functions? Use big-Θ notation.

(a)
```
def belgian_waffle(n):
    i = 0
    total = 0
    while i < n:
        for j in range(50 * n ** 2):
            total += 1
        i += 1
    return total
```

(b)
```
def pancake(n):
    if n == 0 or n == 1:
        return n
    # Flip will always perform three operations and return
       -n.
    return flip(n) + pancake(n - 1) + pancake(n - 1)
```

(c)
```
def toast(n):
    i, j, stack = 0, 0, 0
    while i < n:
        stack += pancake(n)
        i += 1
    while j < n:
        stack += 1
        j += 1
    return stack
```

CS61A Spring 2019: Catherine Cang and Alex Stennet, with
Albert Xu, Albert Yu, Alex Yang, Allan Yu, Amy Hung, Chris Niu, Danelle Nachum, Evan Corriere, Jack Coyle, Jade Singh, Jason Chang, Jemmy Zhou, Kevin Wang, Lucas Pan, Mary Xu, Michelle Cheung, Olivia Shiah, Patricia Ouyang, Richard Roggenkemper, Samy Cherfaoui, Shreya Sahoo, Varun Jhunjhunwalla

2. Consider the following functions:

```
def hailstone(n):
    print(n)
    if n < 2:
        return
    if n % 2 == 0:
        hailstone(n // 2)
    else:
        hailstone((n * 3) + 1)

def fib(n):
   if n < 2:
      return n
   return fib(n - 1) + fib(n - 2)

def foo(n, f):
    return n + f(500)
```

In big-Θ notation, describe the runtime for the following with respect to the input n:

(a) `foo(n, hailstone)`

(b) `foo(n, fib)`

3. **Orders of Growth and Trees:** Assume we are using the ADT tree implementation introduced in discussion. Consider the following function:

```
def word_finder(t, p, word):
    if label(t) == word:
        p -= 1
        if p == 0:
            return True
    for branch in branches(t):
        if word_finder(branch, p, word):
            return True
    return False
```

(a) What does this function do?

(b) If a tree has n total nodes, what is the worst case runtime in big-Θ notation?