Discussion 7

DSC 20, Fall 2023

Meme of the Week

When you finally catch the person that's been writing bad code all the time

Agenda

- Classes
- Lab Review
- Side topic: f strings
- SIde topic: shallow vs deep copies

Functions vs Methods

- Functions are independent code that ingest an object directly
- Methods are code that operate attached to a class

(this is just semantic, I like separating the two)

In [45]:
```
sorted('321') # function
```

Out[45]:
```
['1', '2', '3']
```

In [47]:
```
temp = ['3','2','1']
temp.sort()
temp
```

Out[47]:
```
['1', '2', '3']
```

Classes

Classes aggregate code together into a functional body. A lot of effective python code is written as classes (every time you install a new package, it's basically written as a lot of different classes). For example, recall the pandas dataframe. If you look at the source code for pandas, you'll see that it's a very complicated class definition.

Classes all generally have (at the minimum) a constructor (__init__) function and other object related methods that are used.

In [28]:

```
class phone:
    """
    Class representation of a regular phone.
    """
    pass
```

Constructors

- denoted by __init__ (special function)
- should ingest self as the first parameter (ALWAYS)
- populates instance with attributes

Instances

When an object is created from a class, each individual unit is referred to as an instance.
Think of it as "one of x" (for example, an instance of the iPhone class is "1/my iPhone")

Working with self

- Think of classes as a blueprint and everytime you create an instance of one, you've produced a "physical" manifestation.
- We use the self keyword to interact with THIS specific instance of the class.
- Try to think of 'self' as referencing a **specific, singular** instance of the class. Every method that works with a specific instance's values must be passed in self as an argument.

In [8]:
```python
class phone:
    """
    Class representation of a regular phone.
    """
    def __init__(self, maker, version, owner):
        self.maker = maker
        self.version = version
        self.owner = owner

        self.apps = []
        self.free_memory = 800
```

In [12]:
```python
iphone = phone('Apple', 'XR', 'Nikki')
pixel = phone('Google', 'P3', 'Sailesh')
print(iphone.maker)
print(iphone.owner)
print(iphone.apps)
print()
print(pixel.maker)
print(pixel.owner)
print(pixel.apps)
```

```
Apple
Nikki
[]

Google
Sailesh
[]
```

Instance vs Class variables

Instance variables are attached to instances of a class by keyword self (usually done in the constructor). Class variables are variables attached to the class itself.

In [15]:

```
class phone:
    """
    Class representation of a regular phone.
    """
    id_num = 1 # class variable
    o_type = 'phone' # class variable
    def __init__(self, maker, version, owner):
        self.maker = maker
        self.version = version
        self.owner = owner
        self.id = phone.id_num
        phone.id_num+=1

        self.apps = []
        self.free_memory = 800
```

In [24]:

```
iphone = phone('Apple', 'XR', 'Nikki')
pixel = phone('Google', 'P3', 'Sailesh')
print(iphone.id)
print(iphone.o_type)
print()
print(pixel.id)
print(pixel.o_type)
```

```
3
phone

4
phone
```

(Class) Methods

In [13]:
```
class phone:
    """
    Class representation of a regular phone.
    """
    id_num = 1
    def __init__(self, maker, version, owner):
        self.maker = maker
        self.version = version
        self.owner = owner
        self.id = phone.id_num
        phone.id_num+=1

        self.apps = []
        self.free_memory = 800

    def install_app(self, app, memory):
        if self.free_memory - memory > 0:
            self.apps.append(app)
            self.free_memory -= memory
            return True
        return False
```

In [14]:

```
iphone = phone('Apple', 'XR', 'Nikki')
print(iphone.install_app('duolingo', 600))
print(iphone.install_app('genshin impact', 1000))
print(iphone.apps)
```

```
True
False
['duolingo']
```

Lab Review

Question 1

def q2_01(n):

```
for i in range(abs(20-n) * n):
    print(i)
```

Solution: $O(n^2)$

abs(20-n) -> n - 20 -> O$\left(n * (n - 20)\right)$ = O(n^2)

Question 2

def q2_02(n):

```
dictionary = {}
for num in range(0, 100):
    if num < n:
        dictionary[num] = n - num
```

Solution: $O(1)$

No matter the size of n, the number of iterations is always the same

Question 3

def q2_03(n):

```
t = 1
for i in range(n*n):
    for j in range(i):
        t = t * 2
```

Solution: $O(n^3)$

equivalent to $\sum_{i=1}^{n} i^2 = \frac{n(2n+1)(n+1)}{6} = n^3$

Question 4

def q2_04(n):

```
result = 10
for i in range(0, 300 * n, 3):
    result = result + i
for k in range(n):
    result = result + (2 ** n)
for j in range(n * n * n * n):
    result = result + 5*j

return result
```

Solution: $O(n^4)$

Largest term comes from the final for loop, which is n times itself 4 times

Question 5

def q2_05(n):

```
i = 1
while i < n:
    i = i * 5
    j = n
    while j > 0:
        j = j // 2
```

Solution: $O((log(n))^2)$

nested loops compound their complexity - we have a log complexity loop inside another one, meaning that we get $log(n) * log(n)$

Question 6

def q2_06(n):

```
i = n
for j in range(2 * n):
    while i > 1:
        i = i // 2
    i = n
```

Solution: $O(nlog(n))$

nested loops compound their complexity - we have a log complexity loop inside a linear complexity loop, meaning we get $n * log(n)$

Question 7

def q2_07(lst):

```
for i in range(len(lst)):
    for j in range(len(lst)*10+1):
        if i == 1:
            return True
```

Solution: $O(n)$

the outer loop can only run twice, for value $i = 0, 1$. The inner loop has a time complexity of O(n) and on the second iteration of the outer loop, i=1, which means that when the inner loop tries to run again, it will short-circuit due to the return and condition

Question 8

def q2_08(lst):

```
for item1 in lst:
    for item2 in lst[::-1]:
        for i in range(len(lst)+10, len(lst)-10, -1):
            print(str(item1) + str(item2) + str(i))
```

Solution: $O(n^2)$

inner most for loop is O(1), so we just have a nested for loop

Question 9

def q2_09(n):

```
total = 0
for j in range(n):
    if j**2 > n:
        break
    else:
        total += j
return total
```

Solution: $O(\sqrt{n})$

the break condition is when j^2 is greater than n. This can be rephrased as the loop breaks when $j = \sqrt{n+1}$, meaning that the loop will run for O($\sqrt{n}$) times.

Question 10

def q2_10(n):

```
def q2_10_helper(n):
    return sum([1 for i in range(n) if i%2 == 0])

total = 0
for j in range(n):
    total = total + q2_10_helper(n)
return total
```

Solution: $O(n^2)$

the helper function has a linear time complexity - it takes linear time for the list comp and another linear time to calculate the sum. The for loop that calls the helper function is also linear complexity, resulting in a compounding time complexity again.

f-strings

- strings with embedded expressions
- python can evaluate snippets as subsections of strings
- must be denoted be f and {}'s

In [27]:

```python
start = 30
end = 100

print(f'the program started at {start} seconds and \
completed at {end} seconds')
print(f'the total runtime was {end-start} seconds')
```

```
the program started at 30 seconds and completed at 100 seconds
the total runtime was 70 seconds
```

alternatives

In [51]:

```
print('the program started at {} seconds and \
completed at {} seconds'.format(start,end))
print('the total runtime was {} seconds'.format(end-start))
```

```
the program started at 30 seconds and completed at 100 seconds
the total runtime was 70 seconds
```

In [55]:

```
print('the program started at %s seconds and \
completed at %s seconds' %(start,end))
print('the total runtime was %s seconds' %(end-start))
```

```
the program started at 30 seconds and completed at 100 seconds
the total runtime was 70 seconds
```

Deep vs Shallow Copies

In short, shallow copies copy the **reference** while deep copies copy the **object**. Changes made to a shallow copy are reflected in the original object because the reference is maintained. Changes made to a deep copy are **not reflected** in the original object because it's a completely different object with a difference reference.

pythontutor

In [3]:

```
even_nums = [2,4,6,8,10]
odd_nums = [1,3,5,7,9]

shallow = even_nums
deep = list(odd_nums)

shallow.append(0)
deep.append(0)

print("even_nums: %s" %even_nums)
print("shallow: %s" %shallow)
print("odd_nums: %s" %odd_nums)
print("deep %s" %deep)
```

```
even_nums: [2, 4, 6, 8, 10, 0]
shallow: [2, 4, 6, 8, 10, 0]
odd_nums: [1, 3, 5, 7, 9]
deep [1, 3, 5, 7, 9, 0]
```

This concept extends to all objects, not just lists

In [4]:
```python
class counter:
    def __init__(self, curr_val = 0):
        self.curr_val = curr_val
    def get_counter(self):
        return self.curr_val
    def increment(self, val=1):
        self.curr_val += val
    def decrement(self, val=1):
        self.curr_val -= val
```

In [5]:
```python
c1 = counter()
shallow_c1 = c1
deep_c1 = counter(c1.get_counter())
shallow_c1.decrement(2)
deep_c1.increment(2)
print("c1's value: %s" %c1.get_counter())
print("shallow_c1's value: %s" %shallow_c1.get_counter())
print("deep_c1's value: %s" %deep_c1.get_counter())
```

```
c1's value: -2
shallow_c1's value: -2
deep_c1's value: 2
```

practice questions

This week's questions will be a quiz on gradescope; note that this is purely for you to practice, your discussion participation grade is not contingent on your accuracy.

practice question Solutions

In [22]:

```python
class meal:
    meal_id = 0
    def __init__(self, food_groups, calories, meal_type):
        self.food_groups = food_groups
        self.calories = calories
        self.meal_type = meal_type

        self.meal_id = meal.meal_id
        meal.meal_id += 1

    def add_food(self, food_group, calories):
        self.food_groups.append(food_group)
        self.calories+=calories

m1 = meal(['a'], 100, 'lunch')
m2 = meal(['b'], 700, 'dinner')
print(m1.meal_id)
print(m2.meal_id)
print(meal.meal_id)
```

```
0
1
2
```

In [26]:

```
class SimpleClass:
    def some_method(self):
        print("This is a simple class.")

obj = SimpleClass()
obj.some_method()
```

```
This is a simple class.
```

In [2]:

```python
start = [1,2,3]
x=[1,2,3]
y=x
z=list(x)
x.append(4)
print(f'the starting list is {start}. x = {x}, y = {y}, and z = {z}')
```

```
the starting list is [1, 2, 3]. x = [1, 2, 3, 4], y = [1, 2, 3,
4], and z = [1, 2, 3]
```

In [23]:

```python
# sum of digits
def foo(n):
    if n < 10:
        return n
    else:
        return  n % 10 + foo(n//10)

foo(12345)
```

Out[23]:

```
15
```

Thanks for coming!