Class & objects

COSC 101, 2018-04-16

Announcements
- Homework 8 due Thursday

Outline
- Warm-up
- Programming paradigms
- Creating objects
- Classes
- Return exams

Warm-up
Write a function called nest that takes a list of items and returns a list with each item nested according to its position in the list. For example, nest([1, 2, 3]) should return [1, [2, [3]]] and nest([4]) should return [4]. To help you write the function, first determine: what is the condition for the base case? what is the result for the base case? how will the list be divided in the recursive case?

def nest(L):
    if len(L) == 1:
        return L
    else:
        easier = L[0]
        harder = L[1:]
        harder_nested = nest(harder)
        return [easier, harder_nested]

Programming paradigms

<table>
<thead>
<tr>
<th>Procedural</th>
<th>Object-oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separate code into functions</td>
<td>Group together related functions and the values they manipulate --- object is a set of related values and functions</td>
</tr>
<tr>
<td>Values on which to operate are passed as parameters; values are not “shared” between functions</td>
<td>Values associated with an object are accessible to the object’s functions without needing to pass them as parameters; values are “shared” between functions</td>
</tr>
<tr>
<td>Function call only includes a function’s name E.g., sum</td>
<td>Function call includes the name of an object and the name of a function separated by a dot (.) E.g., myrtle.forward(100)</td>
</tr>
</tbody>
</table>

Object-oriented programming
- Objects have both state (i.e., values assigned to variables) and functions
- Example: Turtle
  - Values: _position, _orient, _pencolor, etc.
  - Functions: forward, right, pencolor, etc.
**Use the dir() function to see all of an object's functions and the vars() function to see all of an object's state variables**

```python
myrtle = Turtle()
print(dir(myrtle))
print(vars(myrtle))
```

**Choose an object in the real world that you may want to represent in a program. --- a car**

a) **What state variables should the object have?**
   - make, model, color, miles per gallon, speed, fuel level, etc.

b) **What functions should the object have?**
   - start, accelerate, brake, shutoff, refill

---

### Creating objects

- **An object is created using a constructor**
  - A constructor is called using the name of the object's type (or class)
    ```python
    myrtle = Turtle()
    ```
  - A constructor may take one or more parameters, which are typically used to set the object's initial state
    ```python
    red_pixel = Pixel(255, 0, 0) # red, green, blue
    ```

- **A program may have multiple objects of the same type**
  - Create another object by calling the constructor again and assigning a different variable name
    ```python
    crush = Turtle()
    ```
  - Each object has its own state values, even if the objects are the same type
    ```python
    blue_pixel = Pixel(0, 0, 255) # red, green, blue
    print(red_pixel.getRed(), red_pixel.getBlue())
    print(blue_pixel.getRed(), blue_pixel.getBlue())
    Output:
    255 0
    0 255
    ```
  - Each object of the same type has the exact same set of functions, but the functions access and modify the state of the object specified in the function call
    ```python
    crush.forward(100)
    print(myrtle._position)
    print(crush._position)
    Output:
    (0.0, 0.0)
    (100.0, 0.0)
    ```

---

### Classes

- **How do we define a new type of object? --- using a class**

- **A class is a template for objects of a particular type**
• Syntax for defining a class
  class CLASSNAME:
    """Docstring describing what the class represents""
    
    def __init__(self):
      """Docstring for the constructor that describes the initial state values""
      self.var1 = "initial value"
      self.var2 = 0
  
• Every class has a name and a constructor
• By convention, the first letter of a class name is capitalized
• The special function name __init__ is used for the constructor
  ○ When you create a new object using the name of the class, it is the special __init__ function
    that is actually called --- e.g.,
    myrtle = Turtle()
    calls the Turtle class's __init__ function (i.e., constructor)
  ○ The constructor always takes a parameter called self that refers to the object being
    constructed
  ○ The object's state variables are accessed using self and the name of the state variable
    separated by a dot (.) --- e.g., self.var1
  ○ The constructor may take additional parameters, which are typically used to provide the initial
    values for an object's state variables
• A class definition may include additional functions as desired
  ○ Functions within a class always take a parameter called self
• Assume you are given the following class definition:
  class Rectangle:
    """A class representing a rectangle""
    
    def __init__(self, w, h):
      self.width = w
      self.height = h

    def area(self):
      return self.width * self.height
  
  a) What is output by the following code?
    rect1 = Rectangle(5, 10)
    print(rect1.area())
    rect2 = Rectangle(6, 8)
    print(rect2.area())
    print(rect1.height, 'x', rect1.width)
    print(rect2.height, 'x', rect2.width)
    Output:
    50
    48
    10 x 5
    8 x 6

  b) Write a perimeter function for the Rectangle class that returns a rectangle's perimeter.
    def perimeter(self):
      return  (self.width + self.height) * 2
Return exams

- Q4: slicing is not a string function; string functions are invoked using the name of a variable containing a string value, a dot, and the name of a function
- Q5b: check the return value from read_data; do not wrap the call to read_data in a call to try/except, because the exception is already handled in read_data