Design Recipe

CPE202

Design Recipe

- You should create a blueprint of your solution before typing your code.
- The design recipe lets you efficiently design a solution.
- The idea is taken from this book:
  - “How To Design Programs” by Matthias Felleisen, et. al.

6 steps of Design Recipe

Step #1: Data Definitions

Step #2: Signature, Purpose Statement, Function Header

Step #3: Test Cases - Write functional examples

Step #4: Template - Write pseudo code with placeholders for values

Step #5: Write the body of the function

Step #6: Test

Example Problem

Write a function **split_list** to split a list of rectangles **rect_list** into two lists: the one containing rectangles whose areas are smaller than a specified value **target** and the other containing rectangles whose areas are larger than the target. Return a tuple of two lists. Rectangles whose areas are the same as the target should not be included in the lists.

Step One: Data Definitions

- The basic question that you're trying to answer is this: "What data types/structures in my programming language will I use to represent the elements of my problem?"

Example Problem: split

- Inventory of data
  - rect_list: Array of Rectangle objects
  - target: An integer value
  - Rectangle: how do you define a rectangle?
  - Return Value: a tuple of two lists

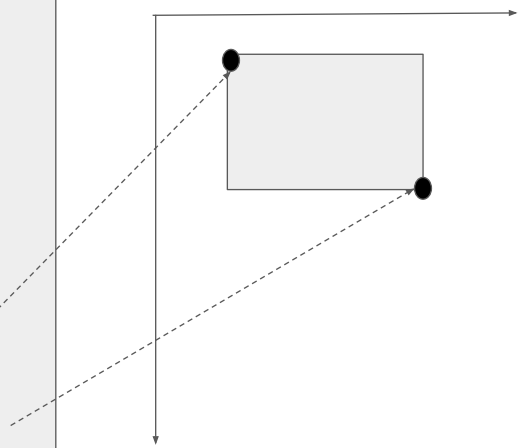
Data Definition

- rect_list (list) : a list of rectangles
- Rectangle : top_left (Point), bot_right (Point)
- Point : x, y

Data Definition

```
class Point:
    """Point class in 2D space
    Attributes:
        x (int) : the x coordinate
        y (int) : the y coordinate
    """

class Rectangle:
    """Rectangle class in 2D space
    Attributes:
        top_left (Point) : the top left corner
        bot_right (Point) : the bottom right corner
    """
```



Step Two: Signature, Purpose Statement, Header

- Function Header
  - Choose meaningful function name and argument names.
  - Do not write the function body yet (Write a dummy for the body).
- Signature describes inputs and outputs of your function
  - describes what kind of values the function accepts, and what kind of value it returns.
- Purpose statement describes the purpose of the function
  - help your co-workers understand what a function is supposed to do.
  - Remind yourself what your function is supposed to do.

Example Problem: split

```
def split(rect_list, target):
    """split a list of rectangles to two lists based on their area
    Args:
        rect_list (list) : a list of Rectanlge objects
        target (int) : the target area value
    Returns:
        list : smaller rectangles
        list : larger rectangles
    """
    pass
```

Step Three: Test cases

- Test cases are functional examples of your solution.
- Identify all kinds of cases your function needs to handle.
  - General cases and Edge cases (Base,Error, and other special cases)
  - If you realize that not all cases can not be solved by your solution, go back to the step #2 or the step #1.

Example Problem: Test for split Function

```
r1 = Rectangle(Point(0, 0), Point(1, 1)) #area 1
r2 = Rectangle(Point(0, 0), Point(2, 3)) #area 6
r3 = Rectangle(Point(0, 0), Point(4, 5)) #area 20
r4 = Rectangle(Point(0, 0), Point(2, 2)) #area 4
r5 = Rectangle(Point(0, 0), Point(1, 2)) #area 2
```

rect_list	target	Return value
[r1, r2, r3, r4, r5]	4	([r1, r5], [r2, r3])
[r1, r2, r3, r4, r5]	-1	([], [r1, r2, r3, r4, r5])
[]	5	([], [])

```
import unittest
from lab1 import get_max

class TestCase(unittest.TestCase):
    def test_get_max(self):
        arr = [1,2,3,4,5]
        self.assertEqual(get_max(arr), 5)

def main():
    # execute unit tests
    unittest.main()

if __name__ == '__main__':
    # execute main() function
    main()
```

Step Four: Template

Write pseudo code

- Add placeholders in the function body
  - Loop
  - Branches
    - Special cases and typical cases
  - Values
    - Fields in objects
  - Helper function

Example Problem: Template for split

```
def split(rect_list, target):
    """split a list of rectangles to two lists based on their area
    Args:
        rect_list (list) : a list of Rectangle objects
        target (int) : the target area value
    Returns:
        list : smaller rectangles
        list : larger rectangles
    """
    #if the list is empty, return empty lists
    #for each rectangle in rect_list
        #area = get_area(rectangle)
        #if the area < target, append it to the smaller list
        #elif the area > target, append it to the larger list
```

Step Five: Write the function body

- Replace pseudo code with actual code.
- Write clean and concise code by following a coding style.
  - Your code needs to be readable so that it is easy to understand, change, and extend.

Python coding style

Python has its recommended coding style guide https://www.python.org/dev/peps/pep-0008/

Google also has published its python style guide http://google.github.io/styleguide/pyguide.html

- Indentation
  - Use 4 spaces per indentation level. (no tabs)
- Maximum line length
  - Limit all lines to a maximum of 79 characters.
  - use indentations to improve readability if code does not fit in one line.
- Blank lines
  - Surround top-level function and class definitions with two blank lines.
- Imports
  - One module per line
    - Bad: import os, sys

Step Six: Test

- Test your function using the test you wrote in the step #3.
- Go back and fix your code if the test fails.
- Sometimes, you have to fix your tests as well.

__main__ — Top-level script environment

- '__main__' is the name of the scope in which top-level code executes.
- A module's __name__ is set equal to '__main__' when read as top-level code (not read as imports from other programs).
- A module can discover whether or not it is running in the main scope by checking its own __name__

Summary

Step #1: Data Definitions

Step #2: Signature, Purpose Statement, Function Header

Step #3: Test Cases - Write functional examples

Step #4: Template - Write pseudo code with placeholders for values

Step #5: Write the body of the function

Step #6: Test