

CS1010S Tutorial 2

Wang Zexin

National University of Singapore
Year 2 QF

wang.zexin@u.nus.edu

January 30, 2017

Today's Agenda

- 1 Recap
- 2 Question One
 - Standard solution
- 3 Question 2
 - Discussion
 - Standard solution
- 4 Question 3
 - Discussion
 - Standard solutions
- 5 Question 4
 - Discussion and Takeaway
 - Standard solutions
- 6 Question 5
 - Discussion and Takeaway
 - Standard solutions
- 7 Extra stuff: one past midterm question and one small game

Recap - Functional Abstraction

What have you learnt?

What is functional abstraction? Explain non-abstractly

- How to write def properly (General form)
 - name
 - formal parameters - input
 - body
 - return type - output
- After that, Prof Leong spent a lot of time on runes.py...
 - You may not even understand what is that!
 - Takeaway: You need to be able to read other people's code
 - and write some more on top of his code
- OK so runes.py
 - primitives building blocks
 - primitive operations (simple functions)
 - derived operations (composite functions)
 - stacking, patterns, overlaying, scaling

Recap - Functional Abstraction

There are more content to recap. . .

- Anonymous function! This can be hard
 - lambda input : output
 - E.g. $\text{lambda } x : x * x$
 - Functions are objects!
 - They can be taken in as parameters
- What is a good abstraction?
 1. Scalable : tasks and subtasks (Divide and conquer)
 2. Maintainable: Comprehensive (Ease your debugging)
 3. Reusable : Capture common patterns
 4. Black box : Provide only relevant details for user
 5. Specification and Implementation
- Approaches to tackle problem
 - Try simple examples
 - Strategize step by step
 - Refine and formulate the problem
 - Top-down approach
 - Bottom-up approach

Recap - Recursion & Iteration

Yes this tutorial covers these as well...

- Recursion

- Divide and conquer
- Recurrence: Similar but smaller problems
- Base case: Problem which can be solved easily ($O(1)$)
- Why is it useful?
- When is it not useful?

- Iteration

- For loop
- Use for loop when there is a iteration variable
- E.g. the index of elements in array
- While loop
- Use while loop when there does not exist a clear iteration variable
- but instead a terminating/continuing signal
- E.g. a flag variable

Question 1 Discussion

Define a function `magnitude` that takes in the coordinates of two points on a plane, $(x1, y1)$ and $(x2, y2)$, as arguments and returns the magnitude of the vector between them.

```
def magnitude(x1, y1, x2, y2):  
    # Returns the magnitude of the vector  
    # between the points (x1, y1) and (x2, y2).
```

- No takeaway, think like a human mathematically and simulate

Question 1 Zexin's solution

This is a mathematical problem.

Recall the definition of magnitude of the vector.

```
def magnitude(x1, y1, x2, y2):  
    # Returns the magnitude of the vector  
    # between the points (x1, y1) and (x2, y2).  
    x = x2 - x1  
    y = y2 - y1  
    return (x**2 + y**2) ** 0.5
```

There is no corner cases. (Why?)

Question 2 Part (a) Discussion

One way of calculating the area of a triangle is using the formula area equals base times height over 2. Define a function `area` that calculates and returns the area of any given triangle using this formula. Decide what arguments it requires as input and what its return value should be.

- $Area = \frac{base \times height}{2}$
- The input is assumed to be base and height (numerical value).
- Your output should be of float type.

Question 2 Part (b) Discussion

Another way of calculating the area of a triangle with sides A, B, C is using the trigonometric ratio sine to get $area = \frac{1}{2} \times A \times B \times \sin AB$, where AB is the included angle between sides A and B. The sin function is provided by the math package. You can call it by using sin after including the line **from math import *** at the top of your Python file. For information on how to use the math package, refer to <http://docs.python.org/3.4/library/math.html> Define a function area2 that calculates and returns the area of any given triangle using this formula. Decide what arguments the function requires as input and what its return value should be.

- $Area = \frac{1}{2} \times A \times B \times \sin AB$
- inputs should be A, B and AB(angle), all of float type
- Your output should be of float type as well.

Question 2 (a) and (b) Zexin's solution

```
def area(base, height):  
    return base * height / 2  
  
from math import *  
  
def area2(A, B, AB):  
    return A * B * sin(AB) / 2
```

Two things to discuss:

- Is the included angle measured in **radian**, or degree?
- What is the difference between `from math import *` and `import math`?

Question 2 Part (c) Discussion

Both functions calculate the same result. Can they be directly substituted for each other? Why?

How to tackle this kind of question?

- $Area = \frac{base \times height}{2}$
- $Area = \frac{1}{2} \times A \times B \times \sin AB$
- How do you decide whether two functions can substitute each other?
- Same storage address? `area == area2`?
- Same functionality? How to compare functionality?

Question 2 Part (c) Solutions

Let's define: if two functions possess the same functionality, for any inputs, they should generate the same outputs. More specifically, if we consider for the functions f and g , the domain is D and the input arguments are vectors, e.g. \underline{x} , while $f(\underline{x})$ and $g(\underline{x})$ stand for their outputs

Same functionality: $\forall \underline{x} \in D, f(\underline{x})$ and $g(\underline{x})$ both exists, and $f(\underline{x}) = g(\underline{x})$

- $Area = \frac{base \times height}{2}$
- $Area = \frac{1}{2} \times A \times B \times \sin AB$
- These two functions take in different **number** of arguments.
- These two functions take in different arguments.
- So, no. They cannot substitute each other.
- One more matter to ponder: what if they do extra things other than returning values? How should we compare their functionality?

Question 2 Part (d) Discussion

We can also calculate the area of triangle using Heron's Formula, $area = \sqrt{s(s-a)(s-b)(s-c)}$ where $s = \frac{a+b+c}{2}$. Assume you are given a function `herons_formula` that takes 3 arguments `a`, `b`, `c` and returns the area of a triangle with sides of length `a`, `b`, `c`.

Define a function `area3` that uses Heron's formula to calculate and return the area of a given triangle given the `x,y` coordinates of the 3 points of the triangle.

You may use the magnitude function defined in Question 1.

- Use the magnitude function to calculate the length of three sides,
- Then use the Heron's formula to calculate,
- Prepare to receive six float inputs for the coordinates of three points,
- Your output value should be of float type.

Question 2 (d) Zexin's solution

```
def herons_formula(a, b, c):  
    s = (a + b + c) / 2  
    return sqrt(s * (s - a) * (s - b) * (s - c))  
  
def area3(x1, y1, x2, y2, x3, y3):  
    oneToTwo = magnitude(x1, y1, x2, y2)  
    twoToThree = magnitude(x2, y2, x3, y3)  
    threeToOne = magnitude(x3, y3, x1, y1)  
    return herons_formula(oneToTwo, twoToThree, threeToOne)
```

- herons_formula is just for your viewing and testing purpose. (optional)
- Camel casing is just for improving readability of code. (optional)
- Of course, this function area3 cannot substitute for the others as well.

Question 3 Discussion

For each of the questions below, what is printed when the expressions are evaluated?

This type of question mostly likely to come out during your test/exam.

What is your strategy to tackle these problems?

- Use your brain and paper just like a Python compiler.
 - Memorise and note down the data types and values of each variable,
 - Execute each command accordingly following the control structure.
 - This method is slow when you encounter multiple function calls or iteration.
- Of course, there is a more advanced way of running programs in your head.
 - You can **objectise** each iteration or function call.
 - Ignore their content, think of what it does to the parameters.
 - Summarise what they can do as brief as possible.
 - **Track** the variables accordingly.

Question 3 Part (a) Discussion

What is printed when the expressions are evaluated?

```
def fool():  
    i = 0  
    result = 0  
    while i < 10:  
        result += i  
        i += 1  
    return result
```

You can use either the primitive or more advanced method.

Question 3 Part (b) Discussion

What is printed when the expressions are evaluated?

```
def foo2():  
    i = 0  
    result = 0  
    while i < 10:  
        if i == 3:  
            break  
        result += i  
        i += 1  
    return result
```

What is the effect of **break** in this while loop?

Question 3 Part (c) Discussion

What is printed when the expressions are evaluated?

```
def bar1():  
    result = 0  
    for i in range(10):  
        result += i  
    return result
```

From which number does this for loop **start** from?

Question 3 Part (d) Discussion

What is printed when the expressions are evaluated?

```
def bar2():  
    result = 0  
    for i in range(10):  
        if i % 3 == 1:  
            continue  
        result += i  
    return result
```

What is the effect of **continue** in this case?

Question 3 Zexin's solutions

- Part (a)
 - Both i and result starts from 0 (same for all parts)
 - During each iteration, add i to result and add one to i
 - Stop when i reaches or bigger than 10
 - Result will be the sum of all integers from 0 to 9 inclusive. (45)
- Part (b)
 - During each iteration, if i equals 3, stop the loop
 - otherwise add i to result and add one to i
 - Stop when i reaches or bigger than 10 (this is never executed)
 - Result will be the sum of all integers from 0 to 2 inclusive. (3)
- Part (c)
 - During each iteration, add i to result and add one to i
 - Stop when i reaches 10
 - Result will be the sum of all integers from 0 to 9 inclusive. (45)
- Part (d)
 - During each iteration, add i to result and add one to i if $i \% 3 \neq 1$
 - Stop when i reaches or bigger than 10
 - Result will be the sum of all integers from 0 to 9 excluding those with remainder 1 when divided by 3. (33)

Question 4 Discussion

Write a function `sum_even_factorials` that finds the sum of the factorials of the even numbers that are less than or equal to n , where $n \geq 1$.

What you need to do is obviously:

- **Loop** from 0 to n (why 0?)
- During each iteration, check if the current number is even
- If yes, add its factorial to the result.

Question 4 Zexin's solution

```
def sum_even_factorials(n):  
    result = 0  
    factorial = 1  
    for i in range(0,n+1):  
        factorial *= i  
        if i % 2 == 0:  
            result += factorial  
    return result
```

One advantage of this solution is that it has the smallest possible time complexity.

Question 5 Discussion

```
def f(g):  
    return g(2)
```

Then we have

```
def square(x):  
    return x ** 2
```

```
>>> f(square)  
4
```

```
>>> f(lambda z: z * (z + 1))  
6
```

What happens if we (perversely) ask the interpreter to evaluate $f(f)$?

- This is an application of high-order functions in Python.
- Let's objectise f first:
- Basically it takes in a function g as parameter
- and return the value evaluated wheng takes 2 as parameter.
- Try out this on square and lambda $z : z \times (z + 1)$

Question 5 Zexin's solution

- $f(f)$ will return $f(2)$
- $f(2)$ takes in 2 as g and will return $g(2)$ which is $2(2)$
- Note that 2 is treated as a **function** here to make a call.
- This will give a `TypeError: 'int' object is not callable`
- One takeaway: you should be able to read error messages with the word 'callable' now.

Extra stuff: one past midterm question

If time permits, we will go through this.

```
def boo(x):  
    return lambda y: x(x(y))  
print(boo(boo)(lambda x:x+1)(5))
```

- What is the output of the print statement?
- Don't be intimidated by the complexity of problem first.
- Let's tackle this problem part by part.

Extra stuff: one past midterm question

This question is of the same type as the previous Question 3.

```
def boo(x):  
    return lambda y: x(x(y))  
print(boo(boo)(lambda x:x+1)(5))
```

Let's tackle the *boo(boo)* part first.

- *boo(boo)* takes in *boo* as parameter.
- It will return $\text{lambda } y : x(x(y))$ with *x* replaced by *boo*.
- So *boo(boo)* is equivalent to $\text{lambda } y : \text{boo}(\text{boo}(y))$,
- which is just repeating *boo* twice for one single argument.

Extra stuff: one past midterm question

```
def boo(x):  
    return lambda y: x(x(y))  
print(boo(boo)(lambda x:x+1)(5))
```

Let's tackle the $\text{boo}(\text{boo})(\text{lambda } x : x + 1)$ part then.

- $\text{boo}(\text{boo})$ basically repeats boo twice for one single argument.
- Let's define a function to simplify $\text{lambda } x : x + 1$, addOne .
- Now we need to evaluate $\text{boo}(\text{boo}(\text{addOne}))$ layer by layer.
- $\text{boo}(\text{addOne})$ will return $\text{lambda } y : \text{addOne}(\text{addOne}(y))$,
- which is just $\text{lambda } y : y + 2$, and let's define it to be addTwo
- Similarly, $\text{boo}(\text{addTwo})$ will be addFour ($\text{lambda } y : y + 4$)

Extra stuff: one past midterm question

```
def boo(x):  
    return lambda y: x(x(y))  
print(boo(boo)(lambda x:x+1)(5))
```

Let's tackle the `boo(boo)(lambda x : x + 1)(5)` part then.

- `boo(boo)(lambda x : x + 1)` basically add four to the input.
- `addFour(5)` will return 9.
- Is this answer correct? Let's test it out!

Extra stuff: one small game

Live programming session

If time permits, we shall go through this.

- Slides + relevant material available at:

`https://github.com/wangzexin/Teaching`

- After the tutorial, if you have further questions:

`wang.zexin@u.nus.edu`

Thank You

wang.zexin@u.nus.edu