CS1010S Tutorial 4

Wang Zexin

National University of Singapore
Year 2 QF

wang.zexin@u.nus.edu

February 11, 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
   - Standard solutions
6. Extra stuff: boxing method and an easy past midterm question

Recap - Abstraction

- Functional abstraction
  - Disregard all the details and only pay attention to functionality
  - High-order functions
- Data abstraction
  - Separate **usage** from **implementation**
  - Implementation: Build up (constructor), get details (accessor)
  - Usage: Calculate details, Arithmetic operations, Predicates, Printers
  - Example: Rational numbers
- General abstraction
  - The client will know and will only know what he need to know.

Recap - Tuple

A new data type learnt: Tuple

Why do you use tuple?

- Allow for multiple storage
  - The ability to store **any** number of elements
  - The ability to store **more** based on a former tuple
- Immutable
  - Cannot alter any single element within a tuple
- Allow for nested tuple
  - accessing the elements : x = ((1,(1,0,(3,)),(2,)),(1,),0)
  - What's x[0][1][2]?
  - Useful for building up **ADT**

Recap - Box and pointer notation

- Equality
  - Identity - is
  - Equivalence - ==
- is
  - same object
  - same storage space (same box)
- ==
  - same contents
  - same values
- What is ('apple', 1, 2, 3) == ('apple', (1,), 2, 3) ?

Recap - Debugging

- Syntax error
  - TypeError (callable)
  - Undeclared variables
- Arithmetic error
  - Division by zero
- Runtime error
  - Infinite loop - use Ctrl(Command) + C
  - Accessing elements outside the range
- Logic error

Question 1

The *Composite Simpson's Rule* is a method of numerical integration. Using Composite Simpson's Rule, the integral of a function f from a to b is approximated as

$$\frac{h}{3}[y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \cdots + y_n]$$

where n is an even integer, $h = \frac{b-a}{n}$, $y_k = f(a + kh)$ and the coefficients of y are 1 for y_0 and y_n, 2 for other even values of k and 4 for odd values of k. (Increasing n increases the accuracy of the approximation.) Define a function that takes as arguments f, a, b, and n and returns the value of the integral, computed using the above Composite Simpson's Rule. Use your function to integrate a cube between 0 and 1 (with $n = 100$ and $n = 1000$).

Question 1 Discussion

- We should define a generic function which can accept the cubic function
- Notice that there is a pattern on how the multipliers 4, 2, 4, 2 are ordered.
- And there is a number n which you should loop from 1 to.
- Hence you are highly recommended to use iteration.
- Also, note that f is inputed in as a parameter.
- This *calc_integral* is a high-order function.

Question 1 Zexin's solution

```
def calc_integral(f, a, b, n):
    h = (b - a) / n # assuming n is even
    result = f(a) + f(b)
    for i in range(1, n):
        result += 2 * (1 + i % 2) * f(a + i * h)
    result *= h / 3
    return result
```

- Takeaway: note how to distinguish between even and odd

Question 2

Write a function `g(k)` that solves the following product using the higher-order function `fold`.

$$g(k) = \prod_{x=0}^{k} (x - (x+1)^2)$$

Note that big-Pi (Π) notation used for product in the same way Sigma (Σ) notation is for `sum`. The code for `fold` is reproduced below for your convenience.

```
def fold(op, f, n):
    if n == 0:
        return f(0)
    return op(f(n), fold(op, f, n-1))
```

Question 2 Discussion

A few things to notice:

- op: should be some multiply function
- f: should be the $x - (x + 1)^2$
- n: should be k

Question 2 Zexin's solution

```
def g(k):
    def f(x):
        return x - (x+1) ** 2
    def times(a, b):
        return a * b
    return fold(times, f, k)
```

- Order of growth for time complexity is?
- Order of growth for space complexity is?

Question 3

Show that `sum` (discussed in lecture) is a special case of a still more general notion called `accumulate` that combines a collection of terms. It uses a general accumulation function, which is the argument `combiner` in the example call below:

$$a_1 = a,\ a_n \leq b$$
$$accumulate(\oplus, base, f, a, next, b):$$
$$(f(a_1) \oplus (f(a_2) \oplus (... \oplus (f(a_n) \oplus base)...)))$$

Write the `accumulate` function and show how `sum` can be defined as a simple call to accumulate. Write using both recursive and iterative approaches.

Question 3 Discussion

- Recursion
  - No difficulty in going from outside to inside
- Iteration
  - Firstly note that we need to go from inside to outside
  - Start from the last number a_n and go to a_1
  - However we do not know what is a_n
  - Is there a way to get the value of a_n first then execute?
  - Problem is that we cannot reverse the function *next* also
  - Simple fix: use a tuple to store all a_i
  - Remember how to build up a tuple?
  - And who did a different approach but pass in coursemology?

Question 3 Zexin's solutions

```
def accumulate(combine, base, f, a, next, b):
    if a > b:
        return base
    else:
        return combine(f(a),
                       accumulate(combine, base, f, next(a), next, b))

def accumulate_iter(combine, base, f, a, next, b):
    aTerms = tuple()
    while a <= b:
        aTerms = (a,) + aTerms
        a = next(a)
    result = base
    for a in aTerms:
        result = combine(f(a), result)
    return result
```

- Order of growth for time complexity is ?
- Order of growth for space complexity is ?

Question 4 Discussion

1. Implementation of line segments in 2D plane
    1. Point: make_point, x_point, y_point, print_point
    2. Segment: make, start, end
    3. Implement midpoint_segment, a usage of Segment
2. Implement Rectangle
3. See if different implementations all work

Question 4 Zexin's solution for Point

```
def make_point(x, y):
    return (x, y)

def x_point(pt):
    return pt[0]

def y_point(pt):
    return pt[1]

def print_point(p):
    print ("(", x_point (p), ",", y_point (p), ")")
```

Here we create the Point using tuple in a way that is **only** known by us.

Question 4 Zexin's solution for Segment

```
def make_segment(start, end):
    return (start, end)

def start_segment(segment):
    return segment[0]

def end_segment(segment):
    return segment[1]

def midpoint_segment(segment):
    x1 = x_point(start_segment(segment))
    y1 = y_point(start_segment(segment))
    x2 = x_point(end_segment(segment))
    y2 = y_point(end_segment(segment))
    x = (x1 + x2) / 2
    y = (y1 + y2) / 2
    return make_point(x, y)
```

Here we create Segment using tuple in a way that is **only** known by us.

Question 4 Zexin's solution for Vector

```
def get_vector(segment):
    x1 = x_point(start_segment(segment))
    y1 = y_point(start_segment(segment))
    x2 = x_point(end_segment(segment))
    y2 = y_point(end_segment(segment))
    return (x2 - x1, y2 - y1)

def get_length_vector(vector):
    return (vector[0] ** 2 + vector[1] ** 2) ** 0.5

def perpendicular_vector(v1, v2):
    return v1[0] * v2[0] + v1[1] * v2[1] == 0
```

Here Vector is an ADT designed **only** for future computation.

Question 4 Zexin's solution for Rectangle

```
def make_rectangle(height, width):
    # Check if this can make a rectangle or not
    # Check if connected
    if x_point(height) != x_point(width) and x_point(height) != y_point(width)
        and y_point(height) != x_point(width) and y_point(height) != y_point(width):
        return "The two segments are not properly connected."
    # Check if perpendicular
    vector_height = get_vector(height)
    vector_width = get_vector(width)
    if not(perpendicular_vector(vector_height, vector_width)):
        return "The two segments are not perpendicular."
    return (height, width)

def height_rect(rect):
    return rect[0]

def width_rect(rect):
    return rect[1]
```

Here we create Rectangle using tuple in a way that is **only** known by us.

Question 4 Zexin's solution for Area and Perimeter

```
def area_rectangle(rect):
    height = height_rect(rect)
    width = width_rect(rect)
    vector_height = get_vector(height)
    vector_width = get_vector(width)
    length_height = get_length_vector(vector_height)
    length_width = get_length_vector(vector_width)
    return length_height * length_width

def perimeter_rectangle(rect):
    height = height_rect(rect)
    width = width_rect(rect)
    vector_height = get_vector(height)
    vector_width = get_vector(width)
    length_height = get_length_vector(vector_height)
    length_width = get_length_vector(vector_width)
    return (length_height + length_width) * 2
```

Does this work for another inplementation of Rectangle?

Let's try out!

Extra stuff: boxing method

If time permits, we will go through this.

- How to get the maximum value or minimum value in a tuple/list?
- Do you need to compare every two values?
- Think about a boxing competition.
- Someone will have to beat the one on the stage
- And the winner will continue to stand on the stage
- Until he is beaten up by another boxer
- This process will continue to go on.
- While those already beaten up cannot go up and challenge.
- Until everyone not on stage is beaten up at least once.
- Then we have the champion(maximum).

Extra stuff: an easy past midterm question

If time permits, we will go through this.

```
x = 5
y = 10
z = 15

u = (x, y, z)
u = (x, y, z) + (x, y)

u[0:1]
```

- What is the output of the last statement?
- This question looks much simpler than the former ones!
- It is about and **only** about tuple!

Extra stuff: an easy past midterm question

```
x = 5
y = 10
z = 15

u = (x, y, z)
u = (x, y, z) + (x, y)

u[0:1]
```

- u is just concatenation of (x, y, z) and (x, y)
- And the answer is just a slicing of u
- Starting from 0, ending at 1 exclusively.
- So it is just $(5,)$

Feedback & more

- 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