CS1010S Tutorial 1

Wang Zεxin

National University of Singapore
Year 2 QF

wang.zexin@u.nus.edu

January 22, 2017

Some incentives before you start

What is programming . . .

- Something that can help us
  - Example 1: Excel VBA button
  - Example 2: Python-tailored replacer/finder
- Something that requires thinking
  - You are going to see more examples throughout the semester
  - One thing: think before you type!
- Something that is simple
  - It can be explained in basic layman terms like climbing steps
  - because they are both done step by step!
- Hopefully you can understand these better by the end of this semester . . .

Some more incentives before you start

How to earn EXP from attending tutorials?

Here are the tutorial EXP guidelines:

- Only showing up - 200
- Minimal participation - 210
- Presents not so well - 220
- Presents clearly / Asked meaningful questions - 230
- Presents answer fully / Brings up interesting topics - 240
- Presents multiple questions -250
- Presents better than tutor - 260

Today's Agenda

1. Recap
2. Question One
   - Discussion
   - Standard solution
3. Question 2
   - Discussion
   - Standard solution
4. Question 3
   - Discussion and Takeaway
   - Standard solutions
5. Question 4
   - Discussion and Takeaway
   - Standard solutions
6. Question 5
   - Discussion
   - Standard solutions
7. Extra stuff: debugging techniques

let's recap first: What have you learnt?

- Control Structures
  - if-elif-else statement
  - while loop
- Data Types
  - int
  - float
  - bool
  - str
  - None
- operators
  - arithmetic operators + - * / ** % //
  - logic operators and or not
  - comparision operators >>= <<= == !=
- others
  - type conversion
  - string slicing/concaternation
  - Difference between assignment and equality?

Question 1

- Below is a sequence of expressions. What is the result printed by the interpreter in response to each expression? Assume that the sequence is to be evaluated in the order in which it is presented. You should determine the answers to this exercise without the help of a computer, and only later check your answers.

```
def square(x):
    return x ** 2
square
square(4)
square(square(square(2)))
f = square
f(2)
def try_f(f):
    return f(3)
try_f(f)
def f(z):
    return z
try_f(f)
```

Question 1 Discussion

What do you think?

One person explain to us your thoughts

- No worries.
- I will help.

Question 1 Zexin's solution

This is a weird question if you place the code in a file and run it.

- square itself does not give any output.
- only if you do this: print(square), it gives the address of the function
- square(4) gives 16 only if you do this: print(square(4))
- square(square(square(2))) gives 256 only if you print it
- f is assigned to be exactly the same function as square
- f(2) gives 4 if you print it
- try_f takes a function as input
- try_f(f) basically returns f(3) which is 9
- now f is re-defined to be a trivial function f(x) = x
- try_f(f) returns f(3) which is 3

Question 1 Takeaway

- Observe that some of the examples shown above are indented and displayed over several lines. The indentation level of statements is significant in Python. As Python functions do not have explicity begin or end, and no curly braces to mark where the function code starts and stops. The only delimiter is a colon(:) and the indentation of the code itself.
- This is what your TA/lecturer wants you to know
- **Indentation** matters a lot!

Question 2 First Part Discussion

First, using if-else, define a function odd(x) that returns True when its integer argument is an odd number and False otherwise.

- The input is assumed to be integer.
- Your output should be of boolean value.

Question 2 Second Part Discussion

Now, without using if-else, define the function new_odd(x) that does the same.

- Same input, same output as above
- Now that you cannot use if-else
- What can you use?
- Hint: deterministic function and stepwise function

Question 2 Zexin's solution

```
def odd(x):
    if x % 2 == 1:
        return True
    else:
        return False

def new_odd(x):
    return bool(x % 2)
```

Simple right?

Question 3 Discussion and Takeaway

Write a function that will return the **number of digits** in an **integer**.

- How do you find the number of digits manually?
  - You count, right?
  - Hey, that's part of the takeaway of this question!
  - For this kind of question which can be 'visualize' and 'simulate', **think like a human** before you think like a machine.
- Of course, there is a mathematical way of checking number of digits
  - If it becomes 0 exactly after n divisions by 10, it has ____ digits.
  - (In decimal representation)
  - So the second part of takeaway is: You can always think **mathematically** to solve problems.
  - but don't forget to check the corner cases!
  - Check for 0 and negative numbers! (why?)
  - Third part of takeaway: Don't forget to **check corner cases**.

Question 3 Zexin's two solutions

```
def numberOfDigits(n):
    if n < 0:
        n = -n
    return len(str(n))

def numberOfDigits(n):
    if n == 0:
        return 1
    elif n < 0:
        n = -n
    count = 0
    while n:
        count += 1
        n //= 10
    return count
```

Why did I use 'while i:' ?

Question 4 Discussion and Takeaway

Define a function that takes three numbers as arguments and returns the sum of the squares of the **two larger** numbers.

- Hey this question looks easy and tedious!
- Let's think of a simple solution for this question!
- Takeaway: simpler solution can save you time during the exam.
- How do I find the two larger numbers easily?
- The most troublesome way is to compare three times . . .
- Instead you can turn the task into finding the _____ number

Question 4 Zexin's solution

```
def sumSquareLargerTwo(a, b, c):
    result = a * a + b * b + c * c
    result -= min(a, b, c) ** 2
    return result
```

How do you prove that there is no corner case?

Question 5 Discussion

Write a function is_leap_year that takes one integer parameter and decides whether it corresponds to a leap year, i.e. the is_leap_year returns True if the input parameter is true, and False otherwise.
Then blablablablabla wikipaedia about leap year...

- Hey this question looks easy and tedious!
- So what do your TA/lecturer want you to practice?
- Obviously it is if-statement
- How do you divide into cases?
- The fewer cases the better!

Question 5 Question Reading and Takeaway

In the Gregorian calendar, the current standard calendar in most of the world, most years that are integer multiples of 4 are leap years. In each leap year, the month of February has 29 days instead of 28. Adding an extra day to the calendar every four years compensates for the fact that a period of 365 days is shorter than a solar year by almost 6 hours. This calendar was first used in 1582.
Some exceptions to this rule are required since the duration of a solar year is slightly less than 365.25 days. Over a period of four centuries, the accumulated error of adding a leap day every four years amounts to about three extra days. The Gregorian Calendar therefore omits 3 leap days every 400 years, omitting February 29 in the 3 century years(integer multiples of 100) that are not also integer multiples of 400. For example, 1600 was a leap year, but 1700, 1800 and 1900 were not. Similarly, 2000 was a leap year, but 2100, 2200, and 2300 will not be. By this rule, the average number of days per year is 365 + 1/4 - 1/100 + 1/400 = 365.2425

- Takeaway: This question tells you how to read a question efficiently

Question 5 Zexin's solution

```
def is_leap_year(year):
    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 == 0:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

Check for corner cases! What are they? (assuming non-negative integer)
One person explain whether this solution works or not

Question 5 Zexin's another solution

```
def is_leap_year(year):
    if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):
        return True
    else:
        return False
```

Check for corner cases! What are they? (assuming non-negative integer)
One person explain whether this solution works or not
Doing this trains your logic statements better =)

Extra stuff: debugging techniques

If time permits, we will go through this.

- What kind of errors do you find hard to debug?
- Those you find hard to understand error messages.
- Those you can understand error message but don't know why you are wrong.
- Those you don't even know how to start.
- Anything else?

Extra stuff: Basic debugging techniques

- First: You should be able to **write out** code with no bug.
- Because that's what you should do during the **exam**.
- So how do you achieve that?
- There are steps to follow:
  - 1. Think about how you can tackle the problem mathematically/manually/machine-like
  - 2. Think about a skeleton of your code (pseudocode)
  - 3. Check carefully if your solution solves all the cases
  - 4. Think about the variables and built-in functions to use
  - 5. Now, stop and check whether all the code in your head is meaningful or not. (If you don't even understand what you are writing, **don't** even write.)
  - 6. Now type out what you want to write
  - 7. Remember to check for the syntax(grammar) errors
  - 8. Remember to check for the arithmetic errors
  - 9. Remember to check for the logic errors
  - 10. If still buggy, search online (hopefully useful)

Extra stuff: little bit more advanced debugging techniques

- These are the skills you may frequently use in this module
- There are steps to follow:
  1. Eyeball your program
  2. Think like a machine to compile and run your program
  3. Insert print statements
  4. at where? At certain points you think your program may fail
  5. print what? Print those variables which you think may be abnormal
  6. Printing helps to locate where your program halted.
  7. It also helps you to trace the variables.
  8. Then you should be able to determine which part is correct or not
  9. If still have buggy code, search online (Stack Overflow)
  10. During test/exam when you have **no** computer, you'll have to eyeball (cannot seek help)
- Disclaimer: These techniques may **not** be useful in software engineering development (when your code is super long). You will need an IDE, devise and document testcases, and automate testing to help you with that. (Irrelevant to this course).

Feedback & more

- Slides + relevant material available at:

  `https://blablablabla.com`

- After the tutorial, if you have further questions:

  `wang.zexin@u.nus.edu`

Yay that's all

Thank You