CS1010S Tutorial 3

Sean Ng

AY2018/19 Sem 2, Week 5

Updated 2019-02-12 at 10:56:51

Materials: pengnam.github.io/1010s

Contact: `seanngpengnam@u.nus.edu`

Feedback From Coursemology

Notes about iteration/recursion

- Iteration: describing a continuous process where there is an invariant (i.e factorial)
- Recursion: describing how a smaller subproblem relates to a larger subproblem that you are solving (i.e. factorial)
- Iteration: typically bottom-up
- Recursion: typically top-down

About completing missions/tutorials

- 1. Do not do contests
- 2. Selectively do sidequests

Complexity Analysis

- Use the correct notation, i.e. $O(f(x))$, e.g. $O(n)$, $O(2^n)$.
    - Incorrect: n, 2^n without the $O(\ )$

Complexity Analysis

- Use the correct notation, i.e. $O(f(x))$, e.g. $O(n)$, $O(2^n)$.
  - Incorrect: n, 2^n without the $O(\ )$
- Drop *coefficients* (a.k.a. constants), not *factors*!
  - e.g. $O(1000n3^n)$ simplified is $O(n3^n)$
  - KhanAcademy: Terms, factors, and coefficients review

Abstraction

- Don't *break* abstraction
  - If a function is provided, use that function!

Abstraction

- Don't *break* abstraction
  - If a function is provided, use that function!
- Why?
  - To hide implementation (irrelevant information)

Abstraction

- Don't *break* abstraction
    - If a function is provided, use that function!
- Why?
    - To hide implementation (irrelevant information)
    - So you don't lose marks

Abstraction

Suppose we have a function `make_lamp` that represents a lamp, and the functions `switch_on` and `switch_off` to turn this lamp on and off. We also have `is_on` to check the state of the lamp.

```python
lamp = make_lamp()

print(is_on(lamp))  # False
switch_on(lamp)
print(is_on(lamp))  # True

if is_on(lamp):
    ...
```

Abstraction

Here's one way of *implementing* `make_lamp`, `switch_on`, and `switch_off`.

```
def make_lamp():
    return 0

def switch_on(lamp):
    return lamp + 1

def is_on(lamp):
    return lamp > 0
```

Abstraction

This would be breaking abstraction. **Don't do this.**

```python
lamp = make_lamp()

print(is_on(lamp))   # False
lamp = lamp + 1      # Turn the lamp on
print(is_on(lamp))   # True

if is_on(lamp):
    ...
```

Do not directly 'mess' with how a thing is represented.

Abstraction

Because, if someone changed the way that a lamp was represented, then what would happen?

```
def make_lamp():
    return False

def switch_on(lamp):
    return True

def is_on(lamp):
    return lamp
```

Higher-Order Functions

- Functions are values
    - Just like strings, integers, floats, etc.

Higher-Order Functions

They can be assigned to variables.

```
def add1(x):
    return x + 1

plus1 = add1

add1(99) == plus1(99)  # True
```

Higher-Order Functions

- Functions are values
    - Just like strings, integers, floats, etc.
- *Functions are values*
    - They can be passed as arguments to a function

Higher-Order Functions

They can be assigned to variables.

```python
def apply(func, num):
    return func(num)

apply(lambda x: x + 1, 99)  # 100
```

Higher-Order Functions

- Functions are values
    - Just like strings, integers, floats, etc.
- *Functions are values*
    - They can be passed as arguments to a function
- **Functions are values**
    - They can be returned from a function

Higher-Order Functions

They can be assigned to variables.

```
def addx(x):
    return lambda i: i + x

add99 = addx(99)
add99(101)  # 200
```

Scoping

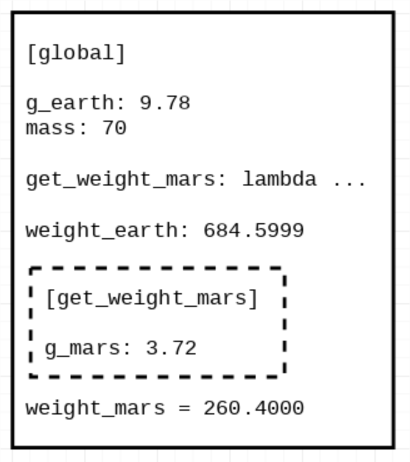
The scope of a variable is a way to describe where you can access (i.e. refer to, or use) the variable.

Scoping

```
g_earth = 9.78
mass = 70

def get_weight_mars():
    g_mars = 3.72
    return 3.72 * mass

weight_earth = g_earth * mass
weight_mars = get_weight_mars()

print(weight_earth)  # 684.5999999999999
print(weight_mars)   # 260.40000000000003
```

(pythontutor.com link)

Scoping



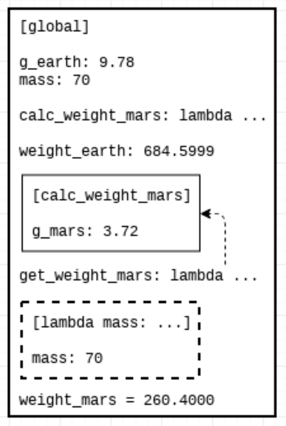
Note: the scope of `[get_weight_mars]` is destroyed after the `return` statement. (This is not always true!)

Scoping

```
g_earth = 9.78
mass = 70

def calc_weight_mars():
    g_mars = 3.72
    return lambda mass: g_mars * mass

weight_earth = g_earth * mass
get_weight_mars = calc_weight_mars()
weight_mars = get_weight_mars(mass)

print(weight_earth)  # 684.5999999999999
print(weight_mars)   # 260.40000000000003
```

(pythontutor.com link)

Scoping



Note: the scope of `[calc_weight_mars]` is *preserved*, 'so that' the `lambda` function can refer to `g_mars`.

Tutorial

Question 1: Coin Change

Draw the tree illustrating the process generated by the `cc(amount, d)` function given in the lecture, in making change for 11 cents.

What are the orders of growth of the space and number of steps used by this process as the amount to be changed increases?

Question 1: Coin Change

```
def cc(amount, kinds_of_coins):
    if amount == 0:
        return 1
    elif (amount < 0) or (kinds_of_coins == 0):
        return 0
    else:
        return cc(amount - first_denomination(kinds_of_coins), kinds_of_coins) +
            cc(amount, kinds_of_coins-1)

def first_denomination(kinds_of_coins):
    '''Returns the corresponding value of the kind of coin, e.g.
        kinds_of_coins: 5 -> 100
        kinds_of_coins: 4 ->  50
        kinds_of_coins: 3 ->  20
        kinds_of_coins: 2 ->  10
        kinds_of_coins: 1 ->   5
    '''
    pass  # pretend it's implemented

def count_change(amount):
    return cc(amount, 5)
```

Question 2: Recursive Function

A function f is defined by the rule that

$$f(n) = \begin{cases} n & \text{if } n < 3 \\ f(n-1) + 2f(n-2) + 3f(n-3) & \text{if } n \geq 3 \end{cases}$$

Write a function `f` that computes f by a *recursive* process.

Question 3: Iterative Function

A function f is defined by the rule that

$$f(n) = \begin{cases} n & \text{if } n < 3 \\ f(n-1) + 2f(n-2) + 3f(n-3) & \text{if } n \geq 3 \end{cases}$$

Write a function `f` that computes f by a *iterative* process.

Question 4: Test for Fibonacci Number

Write a function `is_fib` that returns `True` if `n` is a Fibonacci number, and `False` otherwise.

Question 5: Test for Fibonacci Number

Write a function that returns a function, the latter of which calculates a taxi fare given a distance. Avoid the use of global variables.

Question 5: Test for Fibonacci Number

```
stage1 = 1000
stage2 = 10000
start_fare = 3.0
increment = 0.22
block1 = 400
block2 = 350

def taxi_fare(distance):
    if distance <= stage1:
        return start_fare
    elif distance <= stage2:
        return start_fare + (increment*ceil((distance - stage1) / block1))
    else:
        return taxi_fare(stage2) + (increment*ceil((distance - stage2) / block2))
```