CS131: Python

Discussion 1C: Week 7

Zhiping (Patricia) Xiao

More hints on HW5

- A summary gathering all common questions from the Piazza post (Piazza @230)
- A summary of good test cases with explanation (Piazza @231)
- How to handle the lambda comparison?
  - Determine whether or not the occurred lambda / λ is **shadowed**. If shadowed, don't change its name, and do not treat it as lambda expression anymore.
  - Assume lambda is not shadowed at the top of a list, and thus invalid cases need not be handled (like wrong number of parameters).
  - Comparing a lambda function with a non-lambda function, we can simply do if-else from that level (even not bother changing the lambda to λ).
  - If both expressions are lambda, check if they have the same amount of parameters; if no, do if-else treatment (and remember to lambda->λ); if yes, change variable names and go deeper inside the body recursively comparing everything.
- The above is **just a hint**, not guaranteed to be error-free (and no further detail provided either). Please check carefully with the posts on Piazza.

Getting close to your final

The arrangement of the following weeks’ discussions (including this one) will be:

- Project
- Homework 6
- Final Review

Project: Python

- General-purpose, interpreted language, support OOP
  - One of the most popular languages today
  - Large community --- most of your questions will have solutions online
- Assumption we have: at least basic-level Python background
- We use: Python 3.7
  - On SEASnet be sure to call python**3**
- If you are a started: starter tutorial, official tutorial, W3School and other interactive tutorial
- Deadline: **Nov 25th**
- What: A simple and **paralyzable** proxy in Python
- Requirement
  - **report.pdf**
  - **project.tgz** --- containing AT LEAST **server.py**

Overview of the Project

Build a server herd that can synchronize data and communicate with client applications.

Your prototype should consist of five servers (with server IDs 'Goloman', 'Hands', 'Holiday', 'Welsh', 'Wilkes') that communicate to each other (bidirectionally) with the following pattern:

1. **Goloman talks with Hands, Holiday and Wilkes.**
2. **Hands talks with Wilkes.**
3. **Holiday talks with Welsh and Wilkes.**

Requirement on Report

- Format and length limit: see project spec
- Discuss asyncio pros/cons
  - Is it suitable for this kind of an application?
  - What problems did you run into?
  - Any problems regarding type checking, memory management, multithreading (compare to Java-based approaches)?
  - How does asyncio compare to Node.js?

Review on Basic Python

- Variables are dynamic typing, and no need for declaration before assignment, only letters, numbers and underscores are allowed in names, and names must not start with numbers.
- Program structure is maintained by indentation and colon
- Functions are defined by keyword **def**
- No semicolon needed
- One-line comment by hash-mark / pound-sign **#**
- Multi-line comment by three quotation marks (either single or double)
- Support OOP: **class**
- Does not have a main function by default, executed line by line but functions defined *won't be automatically called* upon running.

More on Python Usage

- Importing your other **XXX.py** files as modules by simply **`import XXX`**
- Directory matters, normally will be solvable by adding **__init__.py** (could be empty), otherwise might need fixation by **`sys`** library

```
import sys
sys.path.append("your_directory_to_import_file_from")
```

- Variables have scope --- local variable with the same name will cast the global definition; if you want it to be the same with global variable, declare it with **`global`** statement.
- Iteration is always low-efficient.
- Run **XXX.py** by **`python XXX.py`** or **`python -i XXX.py`**

More on Python Types

- List are **dynamic** length arrays, stored by reference, mutable. Uses a bit more memory but has O(1) access at random position; also has fast and easy add / remove, etc.
- For your future projects, consider **`numpy`** arrays, that's even better (quick tutorial).
- List comprehension:
  - `[func(x) for x in some_list if condition(x)]`
- Higher-order functions
  - **reduce** is in **`functools`** in Python 3
  - map, filter, reduce
  - Note: the results coming from map / filter

From list comprehension to generator

```
>>> powers = [2**x for x in range(0, 10)]
>>> powers
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
>>> powers = (2**x for x in range(0, 10))
>>> powers
<generator object <genexpr> at 0x111c4c6d0>
>>> next(powers)
1
>>> next(powers)
2
>>> next(powers)
4
>>> next(powers)
8
>>> powers = (2**x for x in range(0, 10) if x > 5)
>>> powers
<generator object <genexpr> at 0x111c4c7d0>
```

Generator in general

```
def batch_generator(dataset, batch_size = 64):
    max_length = len(dataset)
    cursor_index = 0
    while cursor_index < max_length:
        cursor_index += batch_size
        yield dataset[cursor_index - batch_size:cursor_index]

dataset = list(range(1000))
batches = batch_generator(dataset)
print(next(batches))
print(next(batches))
print(next(batches))
```

Python Dictionary

```
>>> my_dict = {1: "integer", "2": {2}, 3: 3, 4: (4,), (5,6):"pair"}
>>> my_dict
{1: 'integer', '2': {2}, 3: 3, 4: (4,), (5, 6): 'pair'}
>>> my_dict[1]
'integer'
>>> my_dict['2']
{2}
>>> my_dict.keys()
dict_keys([1, '2', 3, 4, (5, 6)])
>>> my_dict.values()
dict_values(['integer', {2}, 3, (4,), 'pair'])
>>> my_dict.items()
dict_items([(1, 'integer'), ('2', {2}), (3, 3), (4, (4,)), ((5, 6),
'pair')])
```

Python Tricks - enumerate and zip (and other)

```
>>> for i,k in enumerate(my_dict.keys()):
...     print("the {1}-th key is {0}".format(k,i))
...
the 0-th key is 1
the 1-th key is 2
the 2-th key is 3
the 3-th key is 4
the 4-th key is (5, 6)
>>> my_dict2 = {1:2,3:4,5:6}
>>> reversed_dict = dict(zip(my_dict2.values(),
my_dict2.keys()))
>>> reversed_dict
{2: 1, 4: 3, 6: 5}
```

Can we make a reversed dictionary for my_dict?
Why? (Hint: what kind of values are hashable?)

Asyncio: Asynchronous I/O

Basic concepts related:

- Multi-processing v.s. Multi-threading
- Concurrency v.s. Parallel

Multi-Processing and Multi-Threading

	Multi-Processing	Multi-Threading
# CPUs (Processors) used	Many, but in the same computer system	One, thread is the basic unit of CPU utilization, multiple code segments running **concurrently**
What to share	Computer BUS, memory (optionally: clock, devices, etc.)	The **context** of that process

Parallel and Concurrency

- Parallel: “executed simultaneously”
  - How can we divide a problem into subproblems
  - How can we make the best use of the hardware
- Concurrency: “in progress at the same time”
  - When to execute
  - How to exchange information
  - How to share resources properly
- Parallel $\subset$ Concurrency

Python: Global Interpreter Lock (GIL)

- **What** is it?
  - This is a built-in lock in Python to guarantee only one thread who holds this lock could be executed.
  - **No parallel threading in Python**
    - code with lock & CPU-intensive could be even made slower
  - You may use `multiprocessing` if you **really want to have parallel programming**; you can also write your Cython library where paralleled multi-threading is executed in C/C++, and Python supports these perfectly.
- **Why** Python need this?
  - **In brief: for garbage collection**.
    - Simple garbage collection compared with other methods, thus fast single-threading code
  - Unlike C/C++, Java automatically collect the unused objects for you, and it keeps track of the use of objects by **reference counting**. (If race condition?)
  - Using separate lock for all reference counts?
    - Inefficient and potentially cause deadlocks
  - Other reasons? (Consider how python libraries are implemented? Consider Cython?)

More on GIL

The count returned is generally one higher than you might expect, because it includes the (temporary) reference as an argument to getrefcount().

```
import sys
a = set([1,2,3])
print(sys.getrefcount(a))      # 2
b = a
print(sys.getrefcount(a))      # 3
del b
print(sys.getrefcount(a))      # 2
c = [a, a]
print(sys.getrefcount(a))      # 4
```

What happens if it follows by:

```
c.pop(-1)
print(sys.getrefcount(a))
```

Hint: will garbage collection guaranteed to be happened right after?

Concept Review Exercise: **True or False**

1. **Python does not support threads**
2. **Python does not support parallel programming**

Asyncio: Introduction

- Cooperative multitasking library
  - Tasks can voluntarily take breaks
  - Compare to preemptive multitasking
- Single-threaded approach for concurrent programming
  - Therefore, not parallel
  - Very similar to multithreading, but not the same
- Since Python 3.4 (2014)
  - Still in frequent changes, not that stable API
  - We use 3.7, which is the latest
- Can be used to write TCP clients/servers
  - This is why you have this project

Asyncio Basic Keywords

```
async def g():
    r = await f()
    return r
```

Keyword	Applied	Meaning	Explanation
async	Before defining a function	The function is a coroutine	The function that is a coroutine could suspend itself and give control to any other coroutine.
await	Before the actual execution of a function	Suspending the current coroutine	Suspends the execution of the current coroutine until the awaited function is finished.

Event Loop

- Event loop runs tasks that await for events
- We bind a function to a event loop and then that loop will handle that function.
- **`loop.run_until_complete`** will run that function for once (as is shown in the example code **echo_client.py**)
- **`loop.run_forever`** will repeatedly run that function forever (as is shown in the example code **echo_server.py**)

Resources on Asyncio

- A walkthrough
- Async / Await in Python3.5
  - Mostly still applies to 3.7, but you'd need to carefully check everything out
  - That post explains everything pretty well, a suggested reading linked from our homework spec page
- Documentation of Asyncio
- Another asyncio official documentation, from Python side
- TCP client / server tutorial online
  - And hint code (**echo_client.py, echo_server.py**) based on that
  - Start server, and then start client

Google Map API

- Go visit their website
- Click **Get Started** and follow the steps
- 30 days trial and then it is no longer free (even if it is individual not business account)
- Send Google Server HTTP requests and fetch response via aiohttp