Contents

Chapter 1

Asynchronous Programming

- a style of concurrent programming
- doing many things at once

How does python do multiple things at once

Multiple Processes

- The OS does all the multi-tasking work
- Only option for multi-core concurrency

Multiple Threads

- The OS does all the multi-tasking work
- In CPython, the GIL(Global Interpreter Lock) prevents multi-core concurrency

Asynchronous Programming

- No OS intervention
- One Process, one thread

A style of concurrent programming in which tasks release the CPU during *waiting periods*, so that other *tasks* can use it.

Chapter 2

How is async is implemented

- Async functions need the ability to suspend and resume
- A function that enters a waiting period is suspended, and only resumed when the wait is over
- Four ways to implement suspend/resume in Python
  - Call back function
  - Generator functions
  - Async/await (Python 3.5)
  - Greenlets(require greenlet packages)
- Never pass mutable datatypes to a functions as arguments
- Instance variables and class variables
- static, public and protected variables in python
- decorators # annotations ??
- classmethods
  - (**???**)
  - class methods as alternative constructors
  - take cls as default argument, so mandatory for usage.
  - different from static methods
- static methods
  - (**???**)
  - static methods don't take cls or instance arguments as default argument
- Method resolution order
- Every class inherits from `builtins.object` like `Object()` class in java

Inheriting from super class

- `super().__init__(first, last, pay)`
- `Employee.__init__(self, first, last, pay)`
- `isinstance()` and `issubclass()`
- `threading.Thread.__init__(self)`

First class functions

```
def yell(text):
    return text.upper() + '!'
```

```
>>> yell('hello')
'HELLO!'
```

all data in python programs is represented by objects or relations between objects

```
>>> bark = yell
>>> bark('woof')
'WOOF!'
```

- function objects and their names are two seperate concerns. we can delete the function's original name `yell`
- But another name `bark` still points to the underlying function

```
>>> del yell
>>> yell('hello?')
# will give a NameError

>>> bark('hey')
'HEY!'
```

Chapter 3

Data visualization

- matplotlib
- pandas
- seaborn

jupyter notebooks

- bokeh
- plotly
- D3.js

Chapter 4

Generators

- consider a function to square every number in a list and gives the result in the form of a list.

```
def square_numbers(nums):
    result = []
    for i in nums:
        result.append(i * i)
    return result

my_nums = square_numbers([1, 2, 3, 4, 5])

print my_nums
```

- we can convert the function `square_numbers` into generator. “‘python def square_numbers(nums): for i in nums: yield (i * i)

my_nums = square_numbers([1, 2, 3, 4, 5])

for num in my_nums: print(num) “‘

https://stackoverflow.com/a/38317060/5766752

```
from threading import Thread
import time


def timer(name, delay, repeat):
    print("Timer: " + name + " Started")
    while repeat > 0:
        time.sleep(delay)
        print(name + ": " + str(time.ctime(time.time())))
        repeat -= 1

    print("Timer: " + name + " Completed")


def main():
    t1 = Thread(target=timer, args=("timer1", 1, 5))
    t2 = Thread(target=timer, args=("timer2", 2, 5))
    t1.start()
    t2.start()
```

```
    print("main completed")


if __name__ == "__main__":
    main()
```

Chapter 5

Pygame introduction

- The main steps in a pygame application are :
  1. importing the pygame library
  2. Initialising the pygame library
  3. Creating a window
  4. Initialise game objects
  5. Start the game loop
- `pygame.display.flip()` update the screen every time

function's `__name__` won't affect how you can access it from your code. This identifier is merely a debugging aid. A *variable pointing* to a *function* and the *function* itself are two seperate concerns

there's also `__qualname__` which serves a similar purpose and provides a qualified name string to disambiguate function and class names

Chapter 6

Functions can be stored in data structures

Chapter 7

Functions can be passed to other functions

Chapter 8

Functions can be nested

Chapter 9

Higher order functions

- Functions that can accept other functionss as arguments are also called *higher-order functions*. They are a necessity for the functional programming style.
- The classical example for higher-order functions in python is the built-in `map` function.
- it takes a function and an iterable and calls the function on each element in the iterable yielding the results as it goes along.

```
def yell(text):
    return text.upper() + '!'

>>> list(map(yell, ['hello', 'hey', 'hi']))
['HELLO!', 'HEY!', 'HI!']
```

- map has gone through the entire list and applied the `yell` function to each element
- topological sorting of the dependency graph

Chapter 10

Bottom Up approach

- if we write the program bottom up, the function layout will match the logic flow - it'll go from the fully independent building blocks to the ones that depend on their results.

Chapter 11

top down approach

- for a top-down approach you'd flip the same structure on its head and start with the highest-level building block first, fleshing out the details later.
- vars – function
- key sharing dictionaries
- ordered dicts are not only ordered but also very less size compared to 2.7 dicts
- seperate chaining
- pprint – very helpful for nested datastructures.

Chapter 12

pythonNotes

python notes on important and tricky concepts

Chapter 13

Synchronization in Python

synchronization primitives

- synchronization primitives are used to tell the program to keep the threads in synchrony.
- blocking methods are the methods which block execution of a particular thread until some condition is met.

Locks

- A `Lock` has only two states:
  - locked
  - unlocked
- It is created in the unlocked state and has two principal methods.
  - `acquire()`
  - `release()`

The `acquire()` method locks the `Lock` and blocks the execution until the `release()` method in some other coroutine sets it to unlocked.

Then it locks the `Lock` again and returns `True` The `release()` method should only be called in locked state, it sets the state to unlocked and returns immediately. If `release()` is called in unlocked state, a `RunTimeError` is raised.