Python Beyond Basics Tutorials

Complete each section with activity

Contents

[1: Das Lambda Ordered Queue 1](#_Toc111534632)

[2: Time it ! 2](#_Toc111534633)

[3: Exceptions 3](#_Toc111534634)

[4: Bonus Activity: Async vs Multi vs Subprocesses 4](#_Toc111534635)

# 1: Das Lambda Ordered Queue

**Required Knowledge:**Watch wonderful Lambda Function explainer  
<https://www.youtube.com/watch?v=Ob9rY6PQMfI>

Read instructions on how lambda’s work  
<https://python.swaroopch.com/more.html> (Lambda section)

Activities:  
Restrictions: where possible use map + lambdas as much as possible to reduce size of program, final program should be less than 10 lines of code  
  
Create an application that :

- creates a list of 1000 random numbers

- sort them by size

- print ordered list

# 2: Time it !

**Required Knowledge:**

Watch Magic decorator tutorial from the excellent Corey  
<https://www.youtube.com/watch?v=jCzT9XFZ5bw>

Read instructions on how decorator work  
<https://python.swaroopch.com/more.html>

**Activities:**  
  
1) Write an application that runs for a function that takes a non-small amount of time

\* you can do this any way you wish, I recommend with time.sleep(x) <https://www.programiz.com/python-programming/time/sleep>

2) Run Application and see it take time

3) Create a python decorator to count how long a given function is running for, printing out the time after executing the function

4) Apply this to your function from step1, and see the result

# 3: Exceptions

**Required Knowledge:**Watch Corey’s amazing try/catch explainer  
<https://www.youtube.com/watch?v=NIWwJbo-9_8>

Read instructions on how try/catch work  
<https://python.swaroopch.com/exceptions.html>

**Activities:**  
  
1) Write an application that deliberately raising an error

\* you can do this any way you wish, aka read non-existent file/raise error directly

2) Run Application and see crash message

3) Update your application to run the erroring code in a try/catch block, and make your application print a nice human message instead of a error

# 4: Bonus Activity: Async vs Multi vs Subprocesses

BONUS ACTIVITY, ATTEMPT THIS TIME BOXED TO NO MORE THAN 3-4 HOURS, DO NOT WORRY IF YOU FAIL, IT IS VERY HARD BUT IMPORTANT TO TRY MULTI THREADED CODING

**Required Knowledge:**  
Watch Multiprocessing 101 from “Normalized Nerd”4  
<https://www.youtube.com/watch?v=PcJZeCEEhws>

Watch Threads / multiprocessing 101 video from Dave’s space  
<https://www.youtube.com/watch?v=AZnGRKFUU0c>

Read up on Async  
<https://docs.python.org/3/library/asyncio.html>

Read up on Multi processing / Threading  
<https://www.techopedia.com/definition/24297/multithreading-computer-architecture>  
<https://timber.io/blog/multiprocessing-vs-multithreading-in-python-what-you-need-to-know/>   
<https://docs.python.org/3/library/multiprocessing.html>   
<https://docs.python.org/3/library/threading.html>

**Activities:**

Write 3 separate python programs, each of which to do the same thing, create a folder and create 10,000 files inside that folder (name the files “fib-number-<X>.txt”), and the file should contain that Fibonacci position’s number, aka file “fib-number-5.txt” would contain the 5th fib number “8”

Program 1) “fib-file-sync.py”: All files should be created in a single for loop   
  
time the program’s speed using the time bash command  
$ time fib-file-sync.py

Program 2) “fib-file-async.py”: All files should be divided into 10 loads of a 1000 files, and 10 threads should be created each to do 1000 files each, for example thread one doing fib numbers 1-1000, thread two doing fib numbers 1001-2000, etc  
  
time the program’s speed using the time bash command  
$ time fib-file-async.py

Program 3) “fib-file-multi.py”: All files should be divided into 10 loads of a 1000 files, and 10 processes should be created each to do 1000 files each, for example thread one doing fib numbers 1-1000, thread two doing fib numbers 1001-2000, etc  
  
time the program’s speed using the time bash command  
$ time fib-file-multi.py

Hint:  
you can file a file with this python code

with open('filename.txt', 'w') as filehandle:

filehandle.write('Hello, world!\n')

Lesson:  
single thread very easy  
multi-threads are great at I/O blocked problems, using single core very efficiently but bit harder to program  
multi-processes are great at CPU blocked problems, using multi cores but a lot harder to program  
ps that complexity and difficulty comes from locks, which is a learning topic for yourself when doing your first multi-cpu programming job  
  
  
$ time python fib-file-sync.py

real 0m17.435s

$ time python fib-file-async.py

real 0m11.864s

$ time python fib-file-multi.py

real 0m6.144s