Introduction on

Advanced Python Programming I

Kamal

AptComputingAcademy

Contents

Multi-Processing

Multi-Threading

Cython

Regular expressions

Boto3

- S3
- EC2

Multi-Processing

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Introduction: ()

To run tasks concurrently

Syntax:

Creating child process

Starting child process

Joining child process

Terminate child process

Exchanging data

Using pipe, queue

Synchronization between processes using Lock

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Creating, starting and joining child process:

```
from multiprocessing import Process

import os
def f(name):
    print("pid " ,os.getpid() )
    print('hello', name)

if __name__ == '__main__':
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
    - S3
    - EC2

Using Locks: (Synchronization)

```python
from multiprocessing import Process,Lock
import os

def f(name):
    print("pid " ,os.getpid() )
    print('hello', name)
    for i in range(1000):
        lock.acquire()
        file = open("FileMultiProcessing.txt", 'a+')
        file.write(str(i)+":"+str(os.getpid())+":Hello  world, welcome\n")
        file.close()
        lock.release()
if __name__ == '__main__':
    lock = Lock()
    p1 = Process(target=f, args=('bob',))
    p2 = Process(target=f, args=('bob',))
    p1.start()
    p2.start()
    p1.join()
    p2.join()
```

- **Multi-Processing**
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Terminate child process:

```
if __name__ == '__main__':
    lock = Lock()
    p1 = Process(target=f, args=('bob',))
    p1.start()
    time.sleep(3)
    p1.terminate()
```

Process class attributes:

```
print(p1.name)
print(p1.is_alive())
print(p1.pid)
print(p1.exitcode)
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Using Pipes: (To communicate the data between child and parent):

```
from multiprocessing import Process, Pipe

def f(conn):
    data = conn.recv()
    conn.send('Child: hello parent'+ " "+data)
    conn.close()

if __name__ == '__main__':
    parent_conn, child_conn = Pipe()
    p = Process(target=f, args=(child_conn,))
    p.start()
    parent_conn.send("Hello my dear..")
    print(parent_conn.recv())  # prints "[42, None, ’hello’]"
    p.join()
```

- **Multi-Processing**
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Using Queues: (To communicate the data between child and parent):

```
from multiprocessing import Process, Queue

def f(q):
    q.put([42, True, 'From child...'])

if __name__ == '__main__':
    q = Queue()
    p = Process(target=f, args=(q,))
    p.start()
    print(q.get())
    p.join()
```

- **Multi-Processing**
- Multi-Threading
- Cython
- Regular expressions
- Boto3
    - S3
    - EC2

Using Pool (Multiple processing and loading):

```
from multiprocessing import Pool
import os
def f(x):
    print("Pid = {}, x = {}".format(os.getpid(), x))
    return x*x
if __name__ == '__main__':
    p = Pool(5)
    output = p.map(f,range(100))
    print(output[0])
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

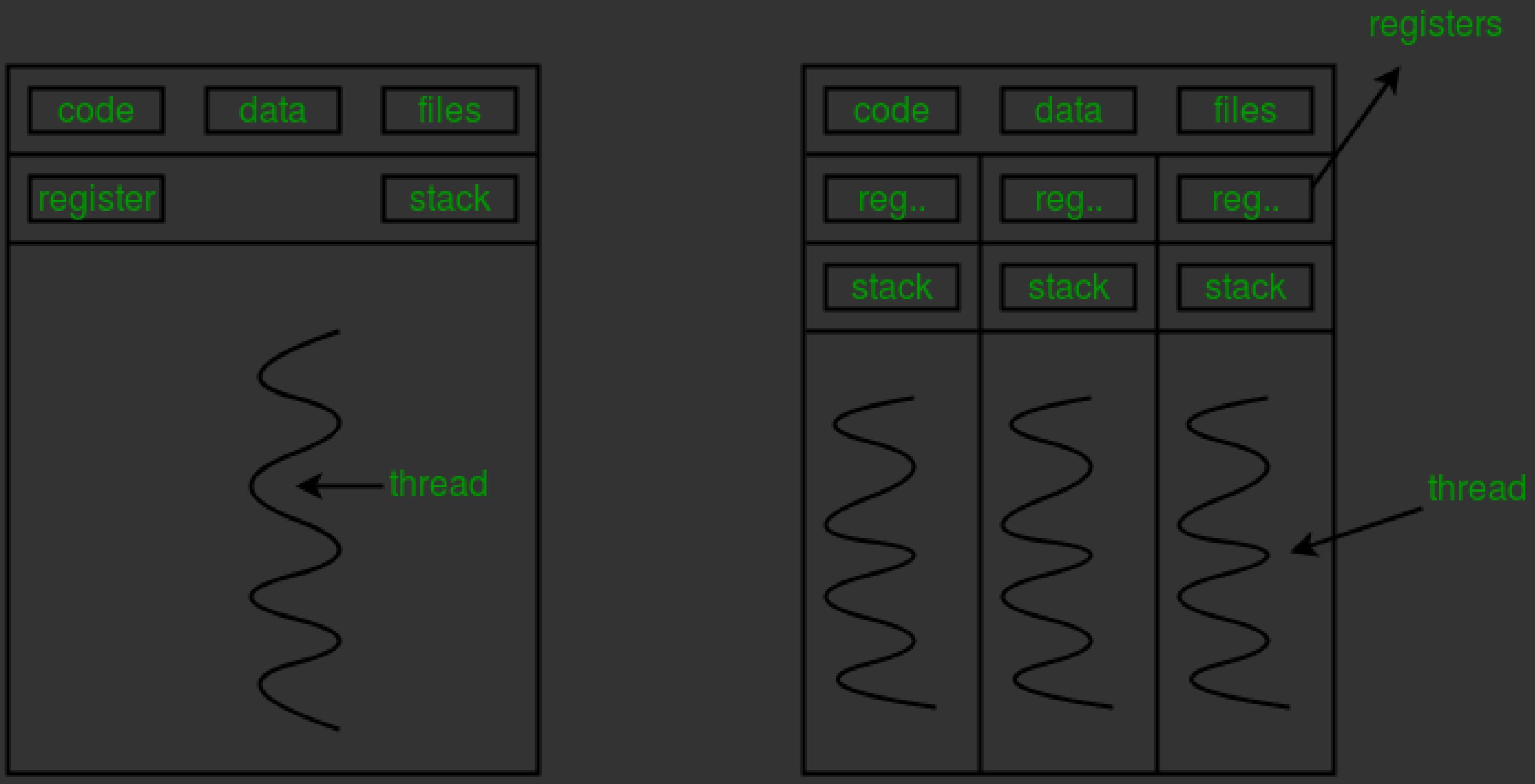
Using Pipes: (To communicate the data between child and parent:

```
if __name__ == '__main__':
    lock = Lock()
    p1 = Process(target=f, args=('bob',))
    p1.start()
    time.sleep(3)
    p1.terminate()
```

Process class attributes:

```
print(p1.name)
print(p1.is_alive())
print(p1.pid)
print(p1.exitcode)
```

Multi-Threading



single-threaded process

multithreaded process

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
    - S3
    - EC2

Cases where should I prefer multi-threads than multi-process??

1. Where inter-thread communication is more
2. Context switch between threads is faster than between processes

Ex:

- Spell checker in MS office
- GUIs which runs back ground in multiple threads
- Chatting/mailing app/website

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Introduction:

To run tasks concurrently

Syntax:

Creating threads

Starting threads

Joining threads

Locking and unlocking

Using Thread methods

- Multi-Processing
- **Multi-Threading**
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Creating threads, starting, joining:

```
import threading
import os
def fun(val):
    print("Val is {}, pid = {} and tid = {}"
         .format(val, os.getpid(),threading.current_thread() ))

tid = threading.Thread(target=fun, args=(10,))
print("In parent pid = {},tid = {}"
    .format(os.getpid(),threading.current_thread()))
tid.start()
print(tid.name)
print()
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Locking and unlocking:

```
import threading
import os
import time
def fun(val):
    print("Waiting for lock aquisition")
    lock.acquire()
    print("Val is {}, pid = {} and tid = {}".format(val, os.getpid(),thread
    lock.release()
tid = threading.Thread(target=fun, args=(10,))
lock = threading.Lock()
lock.acquire()
print("In parent pid = {},tid = {}".format(os.getpid(),threading.current_th
tid.start()
time.sleep(10)
lock.release()
print(tid.name)
print()
```

- Multi-Processing
- **Multi-Threading**
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

threading module functions:

```
threading.activeCount()      ---> Returns no of active threads
threading.currentThread()    ---> Returns current thread name
threading.enumerate()        ---> List of active threads
```

Thread class functions:

```
run()        --> Does not create new thread
isAlive()    --> True/False
getName()    --> Returns thread name
setName()    --> Sets name to the particular thread
```

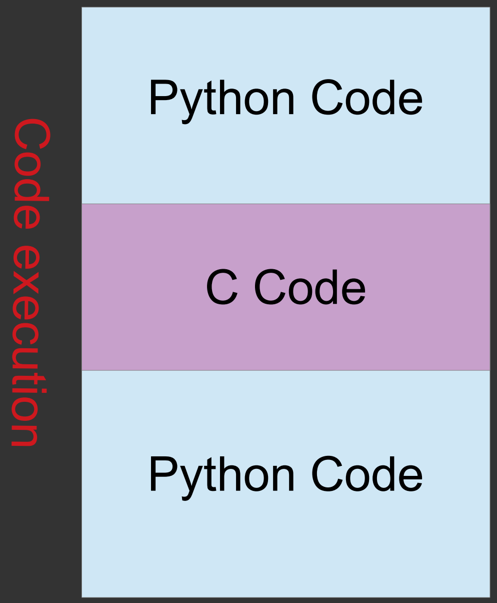
http://docs.cython.org/en/latest/

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Overview:

Purpose of Cython is to get the advantage of C language in Python. It makes writing "C extensions" in Python as easy as Python.

Many modules uses Cython (ex: Numpy module)



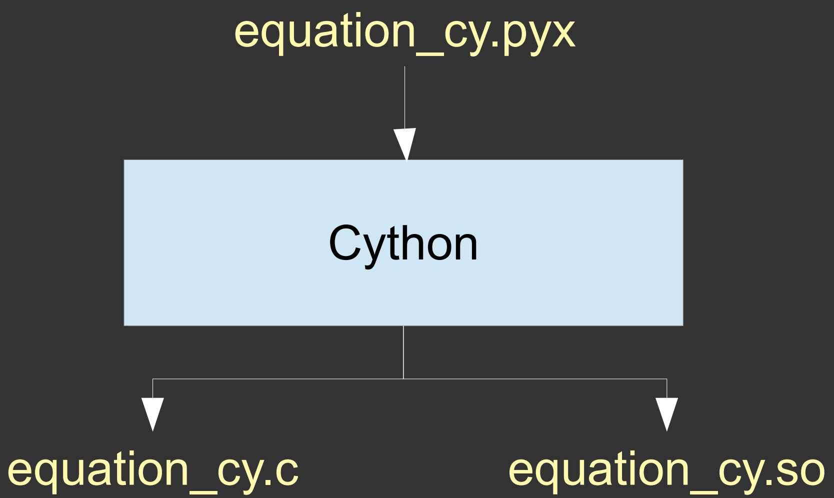
- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Concept behind Cython:

1) Write a plain code in Python (.pyx)with few extra keywords

2) Generate C library (.so) using “Cython”

3) Write main python (.py) code and call the functions/classes written in Cython code.



Sample python program:

```
import equation_cy
Num = 120
Result = equation_cy.equation(num)
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Procedure:

1) Install Cython and gcc
2) Write .pyx file with a function
3) Create setup.py file
4) Generate C file and library
5) Write a main python file and check the performance

- Multi-Processing
- Multi-Threading
- **Cython**
- Regular expressions
- Boto3
  - S3
  - EC2

1) Install Cython and gcc

```
$ pip install cython
$ brew install gcc
```

2) Write .pyx file with a function

```
def equation(n):
    val = (n**3)*10+(n**2)*12+n*24+10
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

3) Create setup.py file

```
from distutils.core import setup
from Cython.Build import cythonize
setup(ext_modules=cythonize("cython_file.pyx"))
```

4) Generate C file and library

```
$ python3 equation_setup.py build_ext --inplace
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

5) Write a main python file and check the performance

```
import equation_cy
import datetime

#def equation(n):
#    val = (n**3)*10+(n**2)*12+n*24+10

start = datetime.datetime.now()
eq_list = list(map(equation_cy.equation, range(1,1000)))
#eq_list = list(map(equation, range(1,1000000)))
end = datetime.datetime.now()
print(end-start)
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Compare the performance using “timeit”:

```
import timeit
cy = timeit.timeit("equation_cy.equation(11)",setup="import equation_cy",number=1000000)
py = timeit.timeit("equation_py.equation(11)",setup="import equation_py",number=1000000)
print("Time = {}".format(cy))
print("Time = {}".format(py))
print("Got x times perf = {}".format(py/cy))
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Significance of def, cdef, cpdef

#def - By Python and Pyx: Data type declaration is not possible
#cpdef - By Python and Pyx: Possible
#cedf - By only Pyx: Possible

Regular expressions

https://www.tutorialspoint.com/python/python_reg_expressions.htm

https://www.ibm.com/support/knowledgecenter/en/SSSH5A_8.0.0/com.ibm.rational.clearquest.schema.ec.doc/topics/sch_pkgs/r_emp_regexpmetachars.htm

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

```
import re
String = '''Mr Kamal Kumar Mukiri 9739858111 kamalbec2004@gmail.com’’’

matcher = re.compile(r'(\n)')

matches = matcher.finditer(String)
for match in matches:
    print(match)
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

PaTtErN: [a-z][A-Z]:

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Meta char	Description
.	Any char (Except new line)
\d	digit
\D	Not a digit
\w	Word char a-z, A-Z, 0-9, _
\W	Not a word char
\s	White space (Space, tab, new line)
\S	Not white space
\b	Word boundary
\B	Not word boudary
Other chars	\n, \t, \r, \v, \f, \z, \Z
(Pattern)	Will be captured which matched with the Pattern

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Meta char	Description
\	Marks the next character as either a special character or a literal.
^	Beginning of the input
$	End of point
*	Preceding character zero or more times
+	One or more
?	Zero or one
x|y	x or y
{n}	Matches exactly n times
{n,}	The o{1,} expression is equivalent to o+ and o{0,} is equivalent to o*.
{n,m}	Match n to m times
[abc]	Any of a,b,cThe o{1,} expression is equivalent to o+ and o{0,} is equivalent to o*.

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

Meta char	Description
[^abc]	Do not match with a,b,c
[a-z]	Match between a to z, all small letters
[^a-z]	Do not match with a to z

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2

import re

```
Mymatch   =    re.match(pattern, string, flags=0)
Mysearch =    re.search(pattern, string, flags=0)
NewString=    re.sub(pattern, repl, string, max=0)
```

“match” : Checks for a match only at the beginning of the string

“search “ : Checks for a match anywhere in the string .

“sub” : Searches for ***patthern*** and replaces with ***repl***

```
Mymatch.group(number=0),group(1),group(2)
Mymatch.groups()
```

“group” : Gives one matched string depending on number

“groups” : Gives the list of matched string

Boto-3

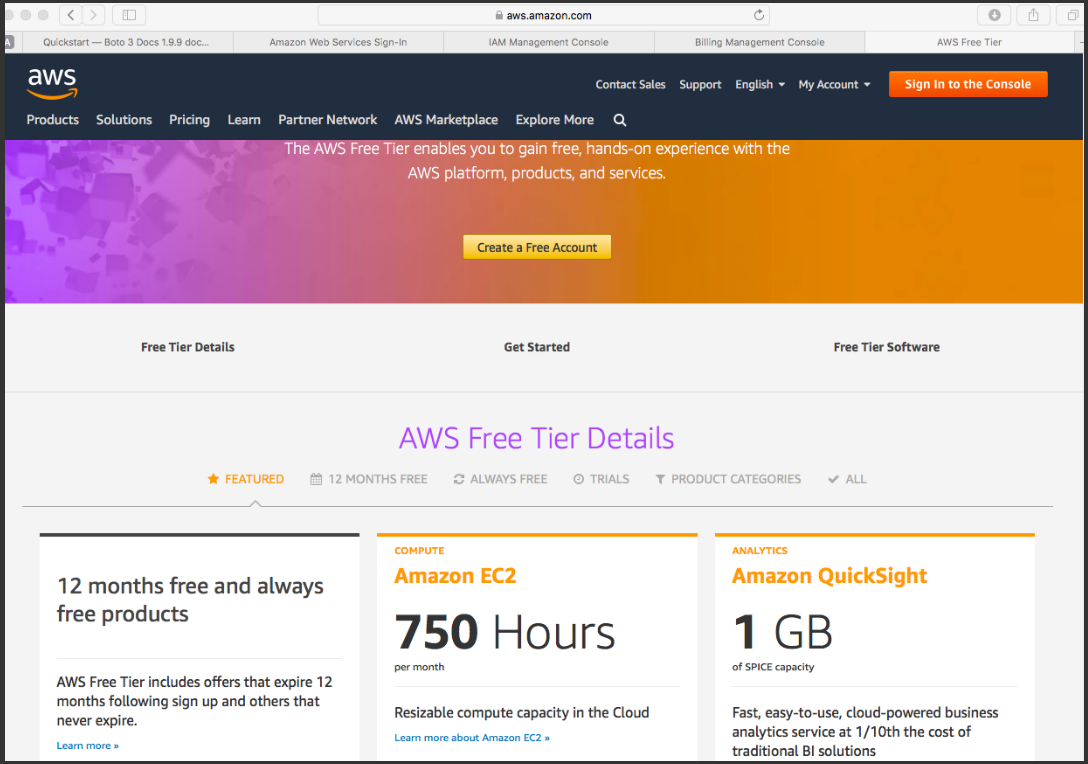
https://boto3.amazonaws.com/v1/documentation/api/latest/guide/examples.html

https://www.youtube.com/watch?v=gVA1FyZejts

https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- **Boto3**
  - S3
  - EC2
  - SQS

Boto is the Amazon Web Services (AWS) SDK for Python, which allows Python developers to write software that makes use of Amazon services like S3, SQS and EC2.



The AWS Free Tier enables you to gain free, hands-on experience with the AWS platform, products, and services.

AWS Free Tier Details

12 months free and always free products

AWS Free Tier includes offers that expire 12 months following sign up and others that never expire.

Learn more »

COMPUTE

Amazon EC2

750 Hours

per month

Resizable compute capacity in the Cloud

Learn more about Amazon EC2 »

ANALYTICS

Amazon QuickSight

1 GB

of SPICE capacity

Fast, easy-to-use, cloud-powered business analytics service at 1/10th the cost of traditional BI solutions

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2
  - SQS

How to AWS using boto3 and PYTHON:

1) Create account in AWS

2) Install “boto3”:

   “$pip install boto3”

3) Install AWS CLI:

   “$brew install awscli”

4) Configure aws :

   ”$aws configure”

   How to get “aws_secret_access_key”?

https://www.cloudberrylab.com/blog/how-to-find-your-aws-access-key-id-and-secret-access-key-and-register-with-cloudberry-s3-explorer/

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2
  - SQS

SQS (Simple Queue Service):
SQS allows you to queue and then process messages.

1) How to create/delete Queue

2) Sending and receiving messages

More information:
https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sqs.html#service-resource

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2
  - SQS

1) How to create/delete Queue

```
import boto3
client = boto3.client('sqs')
q = client.create_queue(QueueName="Sample02")
url = client.get_queue_url(QueueName="Sample02")
print(q)
client.delete_queue(QueueUrl = q['QueueUrl'])
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2
  - SQS

2) Sending and receiving messages

```
import boto3
client = boto3.client('sqs')
q = client.create_queue(QueueName="Sample02")
url = client.get_queue_url(QueueName="Sample02")
print(q)

response = client.send_message(
    QueueUrl=q['QueueUrl'],
    MessageBody='Hello World string')
print("Response is: ",response)

rcvmsg = client.receive_message(QueueUrl=q['QueueUrl'])
print("Received message:", rcvmsg['Messages'][0]['Body'])
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2
  - SQS

S3:

1) Create buckets

2) Delete buckets

3) Upload file

4) Download file

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- **Boto3**
  - **S3**
  - EC2
  - SQS

```
=========== Get list of buckets==============
import boto3

# Create an S3 client
s3 = boto3.client('s3')

# Call S3 to list current buckets
response = s3.list_buckets()

# Get a list of all bucket names from the response
buckets = [bucket['Name'] for bucket in response['Buckets']]

# Print out the bucket list
print("Bucket List: %s" % buckets)
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2
  - SQS

```
=============== Creating bicket ================
import boto3

s3 = boto3.client('s3')
s3.create_bucket(Bucket='my-bucket')
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2
  - SQS

```
================== Uploading file ============
import boto3

# Create an S3 client
s3 = boto3.client('s3')

filename = 'file.txt'
bucket_name = 'my-bucket'

# Uploads the given file using a managed uploader, which will split
up large
# files automatically and upload parts in parallel.
s3.upload_file(filename, bucket_name, filename)
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2
  - SQS

```
================ Downloading file ===========
import boto3
s3 = boto3.resource('s3')
s3.Bucket('mybucket').download_file('hello.txt', '/tmp/hello.txt')
```

- Multi-Processing
- Multi-Threading
- Cython
- Regular expressions
- Boto3
  - S3
  - EC2
  - SQS

Using SQS (Simple Queue Service):

https://boto3.amazonaws.com/v1/documentation/api/latest/guide/sqs.html#sqs

```
import equation_cy
import datetime

#def equation(n):
#    val = (n**3)*10+(n**2)*12+n*24+10

start = datetime.datetime.now()
eq_list = list(map(equation_cy.equation, range(1,1000)))
#eq_list = list(map(equation, range(1,1000000)))
end = datetime.datetime.now()
print(end-start)
```

Thank You……

References:

Maximising Python speed
(http://docs.micropython.org/en/v1.8.6/pyboard/reference/speed_python.html)

Multiprocessing module:
https://docs.python.org/2/library/multiprocessing.html

Write own C code and invoke in Python using Cython:
https://medium.com/@shamir.stav_83310/making-your-c-library-callable-from-python-by-wrapping-it-with-cython-b09db35012a3