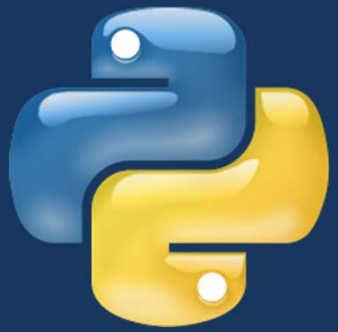
MINDFUL LEARNING

python

A language is not worth knowing unless it teaches you to think differently

- Larry Wall, Randal Schwartz

Professional



python

Multi-threading

Multi-threading

- Running multiple threads is equivalent to running several programs in parallel
- A thread has a beginning, execution sequence and a conclusion
- Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.
- Threads have lesser memory overhead than processes

Threading Module

- To create a thread using threading module
  - Define a new subclass of the *Thread* class.
  - Override the *__init__(self [,args])* method to add additional arguments.
  - Then, override the run(self [,args]) method to implement what the thread should do when started.

MINDFUL LEARNING

python

Example

```
class myThread (threading.Thread):

   def __init__(self, threadID, name, counter):
      threading.Thread.__init__(self)
      self.threadID = threadID
      self.name = name
      self.counter = counter

   def run(self):
      print "Starting " + self.name
      print_time(self.name, self.counter, 5)
      print "Exiting " + self.name


def print_time(threadName, counter, delay):
   while counter:
      if exitFlag:
         threadName.exit()
      time.sleep(delay)
      print "%s: %s" % (threadName, time.ctime(time.time()))
      counter -= 1

# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()
```

MINDFUL LEARNING

python

Supporting Functions

- **run():** The run() method is the entry point for a thread.
- **start():** The start() method starts a thread by calling the run method.
- **join([time]):** The join() waits for threads to terminate.
- **isAlive():** The isAlive() method checks whether a thread is still executing.
- **getName():** The getName() method returns the name of a thread.
- **setName():** The setName() method sets the name of a thread.

Synchronizing Threads

- The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads.
- A new lock is created by calling the *Lock()* method, which returns the new lock.
- The *acquire()* method of the new lock object is used to force threads to run synchronously
- The *release()* method of the new lock object is used to release the lock when it is no longer required

MINDFUL LEARNING

python

Example

```
import threading
import time

class myThread (threading.Thread):
   def __init__(self, threadID, name, counter):
      threading.Thread.__init__(self)
      self.threadID = threadID
      self.name = name
      self.counter = counter
   def run(self):
      print "Starting " + self.name
      # Get lock to synchronize threads
      threadLock.acquire()
      print_time(self.name, self.counter, 3)
      # Free lock to release next thread
      threadLock.release()

def print_time(threadName, delay, counter):
   while counter:
      time.sleep(delay)
      print "%s: %s" % (threadName, time.ctime(time.time()))
      counter -= 1

threadLock = threading.Lock()
threads = []
```

MINDFUL LEARNING

python

Example

```
# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()

# Add threads to thread list
threads.append(thread1)
threads.append(thread2)

# Wait for all threads to complete
for t in threads:
    t.join()
print "Exiting Main Thread"
```

Queues

- The *Queue* module allows you to create a new queue object that can hold a specific number of items. There are following methods to control the Queue −
  - **get():** The get() removes and returns an item from the queue.
  - **put():** The put adds item to a queue.
  - **qsize() :** The qsize() returns the number of items that are currently in the queue.
  - **empty():** The empty( ) returns True if queue is empty; otherwise, False.
  - **full():** the full() returns True if queue is full; otherwise, False.

MINDFUL LEARNING

python

Example

```
import Queue, threading, time

exitFlag = 0

class myThread (threading.Thread):
   def __init__(self, threadID, name, q):
      threading.Thread.__init__(self)
      self.threadID = threadID
      self.name = name
      self.q = q
   def run(self):
      print "Starting " + self.name
      process_data(self.name, self.q)
      print "Exiting " + self.name

def process_data(threadName, q):
   while not exitFlag:
      queueLock.acquire()
         if not workQueue.empty():
            data = q.get()
            queueLock.release()
            print "%s processing %s" % (threadName, data)
         else:
            queueLock.release()
         time.sleep(1)

threadList = ["Thread-1", "Thread-2", "Thread-3"]
nameList = ["One", "Two", "Three", "Four", "Five"]
queueLock = threading.Lock()
workQueue = Queue.Queue(10)
```

MINDFUL LEARNING

python

Example

```
threads = []
threadID = 1

# Create new threads
for tName in threadList:
   thread = myThread(threadID, tName, workQueue)
   thread.start()
   threads.append(thread)
   threadID += 1

# Fill the queue
queueLock.acquire()
for word in nameList:
   workQueue.put(word)
queueLock.release()

# Wait for queue to empty
while not workQueue.empty():
   pass

# Notify threads it's time to exit
exitFlag = 1

# Wait for all threads to complete
for t in threads:
   t.join()
```

Mindful Learning

python

Semaphores

- In programming, especially in Unix systems, semaphores are a technique for coordinating or synchronizing activities in which multiple processes compete for the same operating system resources.

MINDFUL LEARNING

python

Example

```
import threading
sem = threading.Semaphore()

def fun1():
    while True:
        sem.acquire()
        print(1)
        sem.release()

def fun2():
    while True:
        sem.acquire()
        print(2)
        sem.release()

t = threading.Thread(target = fun1)
t.start()
t2 = threading.Thread(target = fun2)
t2.start()
```

Logging Module

- The logging module keeps a record of the events that occur within a program, making it possible to see output related to any of the events that occur throughout the runtime of a piece of software.
- Logs can show you behavior and errors over time, they also can give you a better overall picture of what is going on in your application development process.

Example

MINDFUL LEARNING

python

```
import logging

logging.basicConfig(level=logging.DEBUG)


class Pizza():
    def __init__(self, name, price):
        self.name = name
        self.price = price
        logging.debug("Pizza created: {} (${})".format(self.name,
self.price))

    def make(self, quantity=1):
        logging.debug("Made {} {} pizza(s)".format(quantity, self.name))

    def eat(self, quantity=1):
        logging.debug("Ate {} pizza(s)".format(quantity, self.name))

pizza_01 = Pizza("artichoke", 15)
pizza_01.make()
pizza_01.eat()

pizza_02 = Pizza("margherita", 12)
pizza_02.make(2)
pizza_02.eat()
```

Mindful Learning

python

Example

```
DEBUG:root:Pizza created: artichoke ($15)
DEBUG:root:Made 1 artichoke pizza(s)
DEBUG:root:Ate 1 pizza(s)
DEBUG:root:Pizza created: margherita ($12)
DEBUG:root:Made 2 margherita pizza(s)
DEBUG:root:Ate 1 pizza(s)
```

MINDFUL LEARNING

python

Example #2

```
import logging

logging.basicConfig(
    filename="test.log",
    level=logging.DEBUG,
    format="%(asctime)s:%(levelname)s:%(message)s"
    )


class Pizza():
    def __init__(self, name, price):
        self.name = name
        self.price = price
        logging.debug("Pizza created: {} (${})".format(self.name,
self.price))

    def make(self, quantity=1):
        logging.debug("Made {} {} pizza(s)".format(quantity, self.name))

    def eat(self, quantity=1):
        logging.debug("Ate {} pizza(s)".format(quantity, self.name))

pizza_01 = Pizza("Sicilian", 18)
pizza_01.make(5)
pizza_01.eat(4)

pizza_02 = Pizza("quattro formaggi", 16)
pizza_02.make(2)
pizza_02.eat(2)
```

MINDFUL LEARNING

python

Example #2

```
DEBUG:root:Pizza created: Sicilian ($18)
DEBUG:root:Made 5 Sicilian pizza(s)
DEBUG:root:Ate 4 pizza(s)
DEBUG:root:Pizza created: quattro formaggi ($16)
DEBUG:root:Made 2 quattro formaggi pizza(s)
DEBUG:root:Ate 2 pizza(s)
2017-05-01 16:28:54,593:DEBUG:Pizza created: Sicilian ($18)
2017-05-01 16:28:54,593:DEBUG:Made 5 Sicilian pizza(s)
2017-05-01 16:28:54,593:DEBUG:Ate 4 pizza(s)
2017-05-01 16:28:54,593:DEBUG:Pizza created: quattro formaggi ($16)
2017-05-01 16:28:54,593:DEBUG:Made 2 quattro formaggi pizza(s)
2017-05-01 16:28:54,593:DEBUG:Ate 2 pizza(s)
```

MINDFUL LEARNING

python

Table of Log Levels

Level	Numeric Value	Function	Used to
CRITICAL	50	logging.critical()	Show a serious error, the program may be unable to continue running
ERROR	40	logging.error()	Show a more serious problem
WARNING	30	logging.warning()	Indicate something unexpected happened, or could happen
INFO	20	logging.info()	Confirm that things are working as expected
DEBUG	10	logging.debug()	Diagnose problems, show detailed information