

# Multithreading

- 1 Concurrent Processes  
processes and threads  
life cycle of a thread
- 2 Multithreading in Python  
the thread module  
the Thread class
- 3 Producer/Consumer Relation

MCS 275 Lecture 30  
Programming Tools and File Management  
Jan Verschelde, 29 March 2010

# Multithreading

- 1** Concurrent Processes  
processes and threads  
life cycle of a thread
- 2 Multithreading in Python  
the thread module  
the Thread class
- 3 Producer/Consumer Relation

# Parallel Processing

## processes and threads

At any given time, many processes are running simultaneously on a computer.

The operating system employs *time sharing* to allocate a percentage of the CPU time to each process.

Consider for example the downloading of an audio file. Instead of having to wait till the download is complete, we would like to listen sooner.

Processes have their own memory space, whereas threads share memory and other data. Threads are often called lightweight processes.

A thread is short for *a thread of execution*, it typically consists of one function.

A program with more than one thread is *multithreaded*.

# Parallel Processing

## processes and threads

At any given time, many processes are running simultaneously on a computer.

The operating system employs *time sharing* to allocate a percentage of the CPU time to each process.

Consider for example the downloading of an audio file. Instead of having to wait till the download is complete, we would like to listen sooner.

Processes have their own memory space, whereas threads share memory and other data. Threads are often called lightweight processes.

A thread is short for *a thread of execution*, it typically consists of one function.

A program with more than one thread is *multithreaded*.

# Parallel Processing

## processes and threads

At any given time, many processes are running simultaneously on a computer.

The operating system employs *time sharing* to allocate a percentage of the CPU time to each process.

Consider for example the downloading of an audio file. Instead of having to wait till the download is complete, we would like to listen sooner.

Processes have their own memory space, whereas threads share memory and other data. Threads are often called lightweight processes.

A thread is short for *a thread of execution*, it typically consists of one function.

A program with more than one thread is *multithreaded*.

# Parallel Processing

## processes and threads

At any given time, many processes are running simultaneously on a computer.

The operating system employs *time sharing* to allocate a percentage of the CPU time to each process.

Consider for example the downloading of an audio file. Instead of having to wait till the download is complete, we would like to listen sooner.

Processes have their own memory space, whereas threads share memory and other data. Threads are often called lightweight processes.

A thread is short for *a thread of execution*, it typically consists of one function.

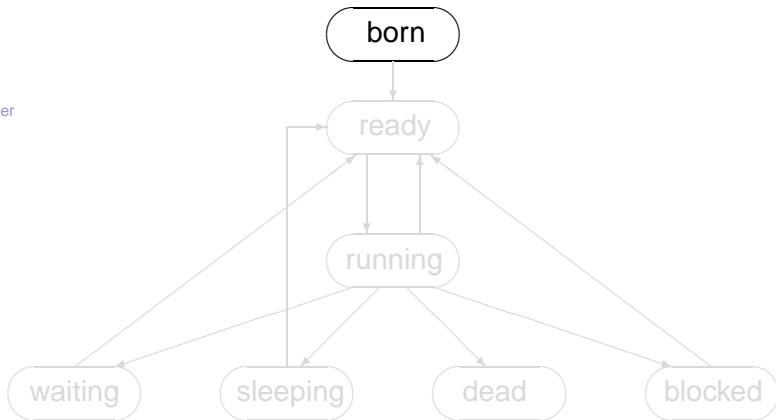
A program with more than one thread is *multithreaded*.

# Multithreading

- 1 Concurrent Processes  
processes and threads  
life cycle of a thread
- 2 Multithreading in Python  
the thread module  
the Thread class
- 3 Producer/Consumer Relation

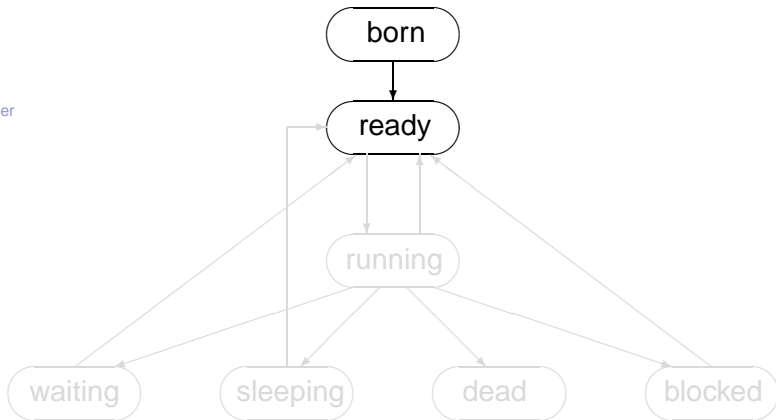
# the Life Cycle of a Thread

a state diagram



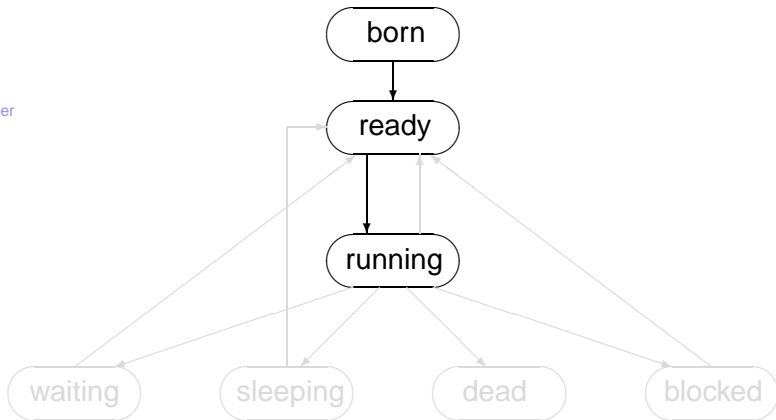
# the Life Cycle of a Thread

a state diagram



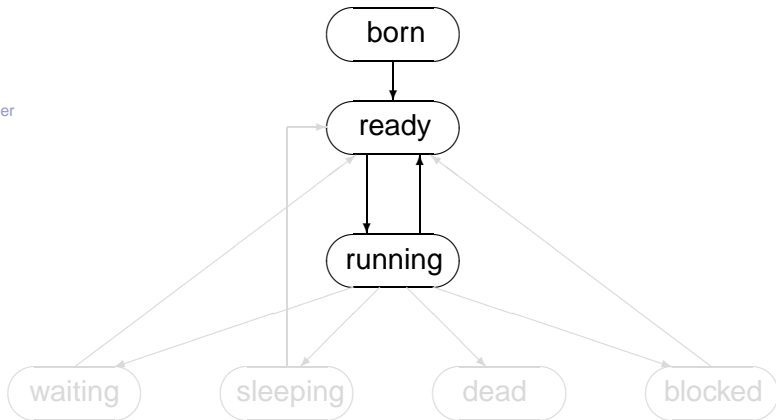
# the Life Cycle of a Thread

a state diagram



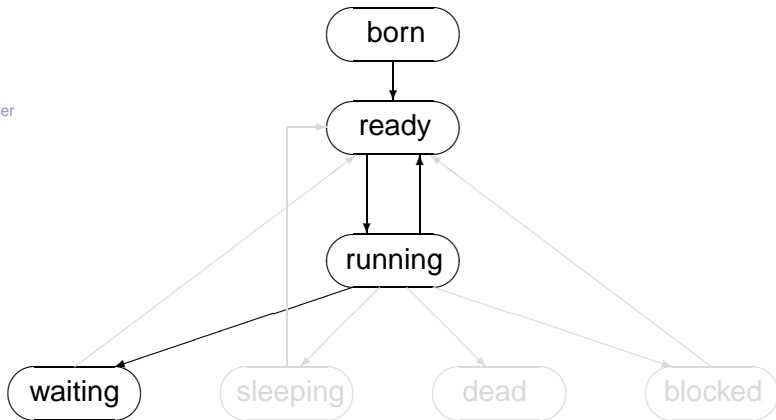
# the Life Cycle of a Thread

a state diagram



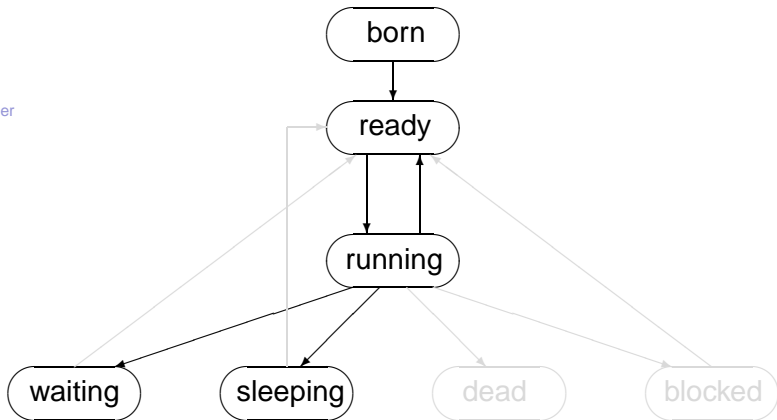
# the Life Cycle of a Thread

a state diagram



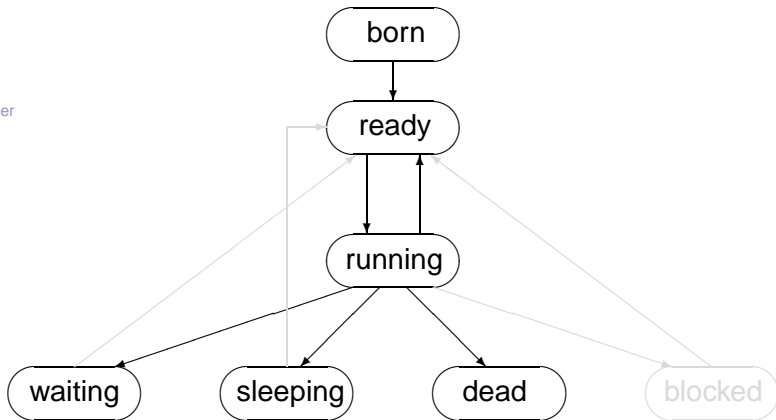
# the Life Cycle of a Thread

a state diagram



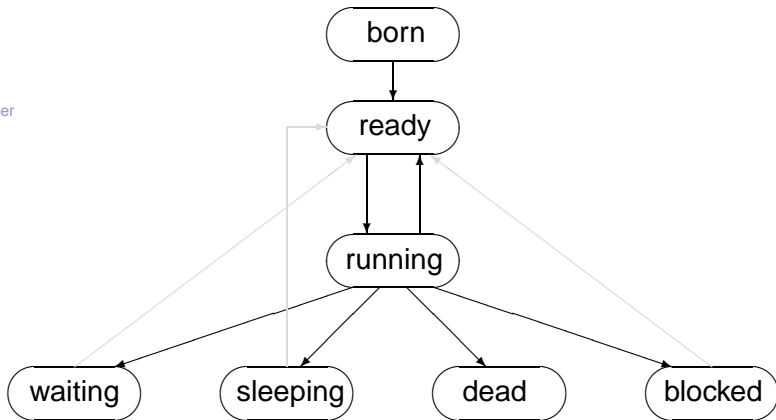
# the Life Cycle of a Thread

a state diagram



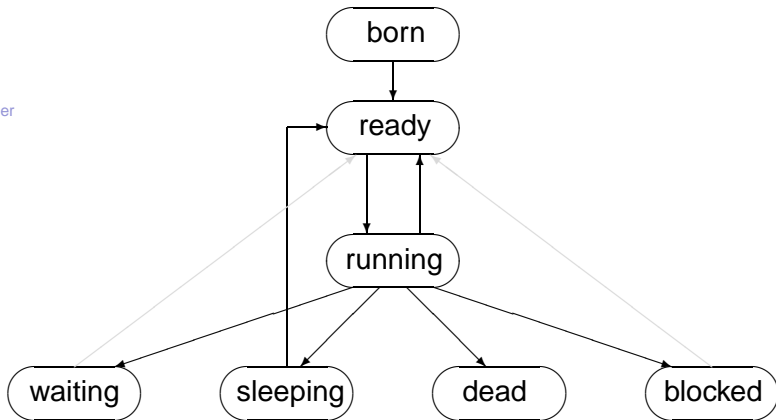
# the Life Cycle of a Thread

a state diagram



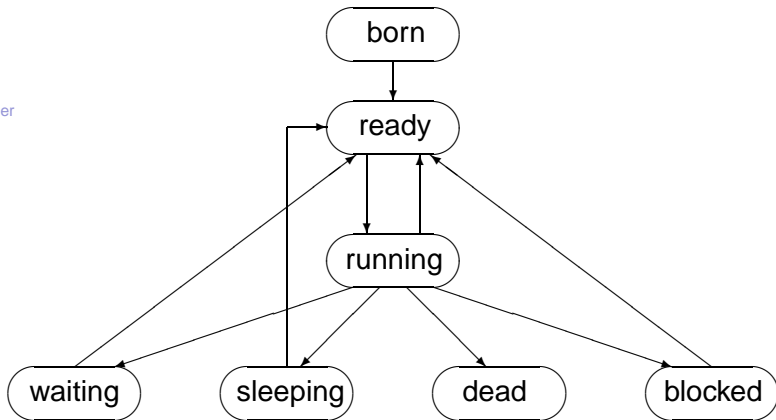
# the Life Cycle of a Thread

a state diagram



# the Life Cycle of a Thread

a state diagram



# Multithreading

- 1 Concurrent Processes  
processes and threads  
life cycle of a thread
- 2 Multithreading in Python  
the thread module  
the Thread class
- 3 Producer/Consumer Relation

# have threads say hello

with the thread module

Our first multithreading Python code will

- 1 import the thread module
- 2 start three threads using  
`thread.start_new_thread`
- 3 each thread will say hello  
and sleep for `n` seconds
- 4 after starting the threads we must wait  
long enough for all threads to finish

# have threads say hello

with the thread module

Our first multithreading Python code will

- 1 import the thread module
- 2 start three threads using  
`thread.start_new_thread`
- 3 each thread will say hello  
and sleep for `n` seconds
- 4 after starting the threads we must wait  
long enough for all threads to finish

# have threads say hello

with the thread module

Our first multithreading Python code will

- 1 import the thread module
- 2 start three threads using  
`thread.start_new_thread`
- 3 each thread will say hello  
and sleep for `n` seconds
- 4 after starting the threads we must wait  
long enough for all threads to finish

# have threads say hello

with the thread module

Our first multithreading Python code will

- 1 import the thread module
- 2 start three threads using  
`thread.start_new_thread`
- 3 each thread will say hello  
and sleep for `n` seconds
- 4 after starting the threads we must wait  
long enough for all threads to finish

# the program hello\_threads.py

using the thread module

```
import thread
import time

def say_hello(name,n):
    "says hello and sleeps n seconds"
    print "hello from " + name
    time.sleep(n)
    print name + " slept %d seconds" % n

print "starting three threads"
thread.start_new_thread(say_hello,("1st thread",3))
thread.start_new_thread(say_hello,("2nd thread",2))
thread.start_new_thread(say_hello,("3rd thread",1))
time.sleep(4) # we must wait for all to finish!
print "done running the threads"
```

# the program `hello_threads.py`

using the `thread` module

Concurrent  
Processes

processes and  
threads

life cycle of a thread

Multithreading  
in Python

the `thread` module  
the `Thread` class

Producer/Consumer  
Relation

```
import thread
import time

def say_hello(name,n):
    "says hello and sleeps n seconds"
    print "hello from " + name
    time.sleep(n)
    print name + " slept %d seconds" % n

print "starting three threads"
thread.start_new_thread(say_hello,("1st thread",3))
thread.start_new_thread(say_hello,("2nd thread",2))
thread.start_new_thread(say_hello,("3rd thread",1))
time.sleep(4) # we must wait for all to finish!
print "done running the threads"
```

# the program `hello_threads.py`

using the `thread` module

Concurrent  
Processes

processes and  
threads

life cycle of a thread

Multithreading  
in Python

the `thread` module  
the `Thread` class

Producer/Consumer  
Relation

```
import thread
import time

def say_hello(name,n):
    "says hello and sleeps n seconds"
    print "hello from " + name
    time.sleep(n)
    print name + " slept %d seconds" % n

print "starting three threads"
thread.start_new_thread(say_hello,("1st thread",3))
thread.start_new_thread(say_hello,("2nd thread",2))
thread.start_new_thread(say_hello,("3rd thread",1))
time.sleep(4) # we must wait for all to finish!
print "done running the threads"
```

the program `hello_threads.py`using the `thread` moduleConcurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe `thread` module  
the `Thread` classProducer/Consumer  
Relation

```
import thread
import time

def say_hello(name,n):
    "says hello and sleeps n seconds"
    print "hello from " + name
    time.sleep(n)
    print name + " slept %d seconds" % n

print "starting three threads"
thread.start_new_thread(say_hello,("1st thread",3))
thread.start_new_thread(say_hello,("2nd thread",2))
thread.start_new_thread(say_hello,("3rd thread",1))
time.sleep(4) # we must wait for all to finish!
print "done running the threads"
```

# running hello\_threads

a screen shot

At the command prompt \$:

```
$ python hello_threads.py
starting three threads
hello from 1st thread
hello from 2nd thread
hello from 3rd thread
3rd thread slept 1 seconds
2nd thread slept 2 seconds
1st thread slept 3 seconds
done running the threads
$
```

# running hello\_threads

a screen shot

At the command prompt \$:

```
$ python hello_threads.py
starting three threads
hello from 1st thread
hello from 2nd thread
hello from 3rd thread

3rd thread slept 1 seconds
2nd thread slept 2 seconds
1st thread slept 3 seconds
done running the threads
$
```

# running hello\_threads

a screen shot

At the command prompt \$:

```
$ python hello_threads.py
starting three threads
hello from 1st thread
hello from 2nd thread
hello from 3rd thread
3rd thread slept 1 seconds
2nd thread slept 2 seconds
1st thread slept 3 seconds
done running the threads
$
```

# running hello\_threads

a screen shot

At the command prompt \$:

```
$ python hello_threads.py
starting three threads
hello from 1st thread
hello from 2nd thread
hello from 3rd thread
3rd thread slept 1 seconds
2nd thread slept 2 seconds
1st thread slept 3 seconds
done running the threads
$
```

# Multithreading

- 1 Concurrent Processes  
processes and threads  
life cycle of a thread
- 2 Multithreading in Python  
the thread module  
**the Thread class**
- 3 Producer/Consumer Relation

# using the `Thread` class

an object oriented approach

The `threading` module exports the `Thread` class.

We create new threads by inheriting from `threading.Thread`, overriding the `__init__` and `run`.

After creating a thread object, a new thread is born.

With `run`, we start the thread.

Main difference with the `thread` module is the explicit difference between the born and running state.

# using the `Thread` class

an object oriented approach

The `threading` module exports the `Thread` class.

We create new threads by inheriting from `threading.Thread`, overriding the `__init__` and `run`.

After creating a thread object, a new thread is born.

With `run`, we start the thread.

Main difference with the `thread` module is the explicit difference between the born and running state.

# using the `Thread` class

an object oriented approach

The `threading` module exports the `Thread` class.

We create new threads by inheriting from `threading.Thread`, overriding the `__init__` and `run`.

After creating a thread object, a new thread is born.

With `run`, we start the thread.

Main difference with the `thread` module is the explicit difference between the born and running state.

# using the `Thread` class

an object oriented approach

The `threading` module exports the `Thread` class.

We create new threads by inheriting from `threading.Thread`, overriding the `__init__` and `run`.

After creating a thread object, a new thread is born.

With `run`, we start the thread.

Main difference with the `thread` module is the explicit difference between the born and running state.

# using the Thread class

an object oriented approach

The `threading` module exports the `Thread` class.

We create new threads by inheriting from `threading.Thread`, overriding the `__init__` and `run`.

After creating a thread object, a new thread is born.

With `run`, we start the thread.

Main difference with the `thread` module is the explicit difference between the born and running state.

# running hello\_threading

## Concurrent Processes

processes and threads

life cycle of a thread

## Multithreading in Python

the thread module

the Thread class

## Producer/Consumer Relation

At the command prompt \$:

```
$ python hello_threading.py
first thread is born
second thread is born
third thread is born
starting threads
hello from first thread
hello from second thread
hello from third thread
threads started
third thread slept 1 seconds
second thread slept 4 seconds
first thread slept 5 seconds
$
```

# running hello\_threading

## Concurrent Processes

processes and threads

life cycle of a thread

## Multithreading in Python

the thread module

the Thread class

## Producer/Consumer Relation

At the command prompt \$:

```
$ python hello_threading.py
first thread is born
second thread is born
third thread is born
starting threads
hello from first thread
hello from second thread
hello from third thread
threads started

third thread slept 1 seconds
second thread slept 4 seconds
first thread slept 5 seconds
$
```

## running hello\_threading

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Python

the thread module

the Thread class

Producer/Consumer  
Relation

At the command prompt \$:

```
$ python hello_threading.py
first thread is born
second thread is born
third thread is born
starting threads
hello from first thread
hello from second thread
hello from third thread
threads started
third thread slept 1 seconds
second thread slept 4 seconds
first thread slept 5 seconds
$
```

# the Class `HelloThread`

its structure

```
import threading
import time
import random

class HelloThread(threading.Thread):
    """
    hello world with threads
    """
    def __init__(self,t):
        "initializes thread with name t"

    def run(self):
        "says hello and sleeps awhile"

def main():
    "starts three threads"

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

# the Class HelloThread

its structure

```
import threading
import time
import random

class HelloThread(threading.Thread):
    """
    hello world with threads
    """
    def __init__(self,t):
        "initializes thread with name t"

    def run(self):
        "says hello and sleeps awhile"

def main():
    "starts three threads"

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

# the Class HelloThread

its structure

```
import threading
import time
import random

class HelloThread(threading.Thread):
    """
    hello world with threads
    """
    def __init__(self,t):
        "initializes thread with name t"

    def run(self):
        "says hello and sleeps awhile"

def main():
    "starts three threads"

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

# the Class HelloThread

its structure

```
import threading
import time
import random

class HelloThread(threading.Thread):
    """
    hello world with threads
    """
    def __init__(self,t):
        "initializes thread with name t"

    def run(self):
        "says hello and sleeps awhile"

def main():
    "starts three threads"

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

# the Class HelloThread

its structure

```
import threading
import time
import random

class HelloThread(threading.Thread):
    """
    hello world with threads
    """
    def __init__(self,t):
        "initializes thread with name t"

    def run(self):
        "says hello and sleeps awhile"

def main():
    "starts three threads"

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

# the Methods of `HelloThread`

constructor and `run`

```
def __init__(self,t):
    "initializes thread with name t"
    threading.Thread.__init__(self,name=t)
    print t + " is born "

def run(self):
    "says hello and sleeps awhile"
    t = self.getName()
    print "hello from " + t
    r = random.randint(1,6)
    time.sleep(r)
    print t + " slept %d seconds" % r
```

# the Methods of HelloThread

constructor and run

```
def __init__(self,t):
    "initializes thread with name t"
    threading.Thread.__init__(self,name=t)
    print t + " is born "

def run(self):
    "says hello and sleeps awhile"
    t = self.getName()
    print "hello from " + t
    r = random.randint(1,6)
    time.sleep(r)
    print t + " slept %d seconds" % r
```

# the main function

## using `HelloThread`

```
def main():  
    "starts three threads"  
    t1 = HelloThread("first thread")  
    t2 = HelloThread("second thread")  
    t3 = HelloThread("third thread")  
    print "starting threads"  
    t1.start()  
    t2.start()  
    t3.start()  
    print "threads started"
```

# the main function

## using `HelloThread`

```
def main():  
    "starts three threads"  
    t1 = HelloThread("first thread")  
    t2 = HelloThread("second thread")  
    t3 = HelloThread("third thread")  
    print "starting threads"  
    t1.start()  
    t2.start()  
    t3.start()  
    print "threads started"
```

# Producer/Consumer Relation

with threads

A very common relation between two threads is that of producer and consumer. For example, the downloading of an audio file is production, while listening is consumption.

Our producer/consumer relation with threads uses

- an object of the class `Producer` is a thread that will append to a queue consecutive integers in a given range and at a given pace;
- an object of the class `Consumer` is a thread that will pop integers from the queue and print them, at a given pace.

If the pace of the produces is slower than the pace of the consumer, then the consumer will wait.

# Producer/Consumer Relation

with threads

A very common relation between two threads is that of producer and consumer. For example, the downloading of an audio file is production, while listening is consumption.

Our producer/consumer relation with threads uses

- an object of the class `Producer` is a thread that will append to a queue consecutive integers in a given range and at a given pace;
- an object of the class `Consumer` is a thread that will pop integers from the queue and print them, at a given pace.

If the pace of the produces is slower than the pace of the consumer, then the consumer will wait.

# Producer/Consumer Relation

with threads

A very common relation between two threads is that of producer and consumer. For example, the downloading of an audio file is production, while listening is consumption.

Our producer/consumer relation with threads uses

- an object of the class `Producer` is a thread that will append to a queue consecutive integers in a given range and at a given pace;
- an object of the class `Consumer` is a thread that will pop integers from the queue and print them, at a given pace.

If the pace of the produces is slower than the pace of the consumer, then the consumer will wait.

# Producer/Consumer Relation

with threads

A very common relation between two threads is that of producer and consumer. For example, the downloading of an audio file is production, while listening is consumption.

Our producer/consumer relation with threads uses

- an object of the class `Producer` is a thread that will append to a queue consecutive integers in a given range and at a given pace;
- an object of the class `Consumer` is a thread that will pop integers from the queue and print them, at a given pace.

If the pace of the produces is slower than the pace of the consumer, then the consumer will wait.

# Producer/Consumer Relation

with threads

A very common relation between two threads is that of producer and consumer. For example, the downloading of an audio file is production, while listening is consumption.

Our producer/consumer relation with threads uses

- an object of the class `Producer` is a thread that will append to a queue consecutive integers in a given range and at a given pace;
- an object of the class `Consumer` is a thread that will pop integers from the queue and print them, at a given pace.

If the pace of the produces is slower than the pace of the consumer, then the consumer will wait.

# Producer/Consumer Relation

with threads

A very common relation between two threads is that of producer and consumer. For example, the downloading of an audio file is production, while listening is consumption.

Our producer/consumer relation with threads uses

- an object of the class `Producer` is a thread that will append to a queue consecutive integers in a given range and at a given pace;
- an object of the class `Consumer` is a thread that will pop integers from the queue and print them, at a given pace.

If the pace of the produces is slower than the pace of the consumer, then the consumer will wait.

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processes

processes and  
threads  
life cycle of a thread

Multithreading  
in Python

the thread module  
the Thread class

Producer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

## running the code

Concurrent  
Processesprocesses and  
threads

life cycle of a thread

Multithreading  
in Pythonthe thread module  
the Thread classProducer/Consumer  
Relation

```
$ python prodcons.py
producer starts...
producer sleeps 1 seconds
consumption starts...
consumer sleeps 1 seconds
appending 1 to queue
producer sleeps 4 seconds
popped 1 from queue
consumer sleeps 1 seconds
wait a second...
wait a second...
wait a second...
appending 2 to queue
producer sleeps 2 seconds
popped 2 from queue
consumer sleeps 1 seconds
wait a second...
appending 3 to queue
production terminated
popped 3 from queue
consumption terminated
```

# Structure of Producer Class

parameters and methods

```
import threading
import random
import time

class Producer(threading.Thread):
    """
    Appends integers to a queue.
    """
    def __init__(self,t,q,a,b,p):
        "thread t to add integers in [a,b] to q"

    def run(self):
        "produces integers at some pace"
```

# Structure of Producer Class

parameters and methods

```
import threading
import random
import time

class Producer(threading.Thread):
    """
    Appends integers to a queue.
    """
    def __init__(self, t, q, a, b, p):
        "thread t to add integers in [a,b] to q"

    def run(self):
        "produces integers at some pace"
```

# Structure of Producer Class

parameters and methods

```
import threading
import random
import time

class Producer(threading.Thread):
    """
    Appends integers to a queue.
    """
    def __init__(self, t, q, a, b, p):
        "thread t to add integers in [a,b] to q"

    def run(self):
        "produces integers at some pace"
```

# Structure of Producer Class

parameters and methods

```
import threading
import random
import time

class Producer(threading.Thread):
    """
    Appends integers to a queue.
    """
    def __init__(self, t, q, a, b, p):
        "thread t to add integers in [a,b] to q"

    def run(self):
        "produces integers at some pace"
```

# Methods of Producer Class

constructor

```
def __init__(self,t,q,a,b,p):  
    """  
    Thread t to add integers in [a,b] to q  
    sleeping between 1 and p seconds.  
    """  
    threading.Thread.__init__(self,name=t)  
    self.queue = q  
    self.begin = a  
    self.end = b  
    self.pace = p
```

# Methods of Producer Class

constructor

```
def __init__(self,t,q,a,b,p):
    """
    Thread t to add integers in [a,b] to q
    sleeping between 1 and p seconds.
    """
    threading.Thread.__init__(self,name=t)
    self.queue = q
    self.begin = a
    self.end = b
    self.pace = p
```

# The Production Method

of class Producer

```
def run(self):  
    "produces integers at some pace"  
    print self.getName() + " starts..."  
    for i in range(self.begin,self.end+1):  
        r = random.randint(1,self.pace)  
        print self.getName() + \  
            " sleeps %d seconds" % r  
        time.sleep(r)  
        print "appending %d to queue" % i  
        self.queue.append(i)  
    print "production terminated"
```

# The Production Method

of class Producer

```
def run(self):
    "produces integers at some pace"
    print self.getName() + " starts..."
    for i in range(self.begin,self.end+1):
        r = random.randint(1,self.pace)
        print self.getName() + \
            " sleeps %d seconds" % r
        time.sleep(r)
        print "appending %d to queue" % i
        self.queue.append(i)
    print "production terminated"
```

# The Production Method

of class Producer

```
def run(self):
    "produces integers at some pace"
    print self.getName() + " starts..."
    for i in range(self.begin,self.end+1):
        r = random.randint(1,self.pace)
        print self.getName() + \
            " sleeps %d seconds" % r
        time.sleep(r)
        print "appending %d to queue" % i
        self.queue.append(i)
    print "production terminated"
```

# The Production Method

of class Producer

```
def run(self):
    "produces integers at some pace"
    print self.getName() + " starts..."
    for i in range(self.begin,self.end+1):
        r = random.randint(1,self.pace)
        print self.getName() + \
            " sleeps %d seconds" % r
        time.sleep(r)
        print "appending %d to queue" % i
        self.queue.append(i)
    print "production terminated"
```

# Structure of Consumer Class

constructor and run

```
import threading
import random
import time

class Consumer(threading.Thread):
    """
    Pops integers from a queue.
    """
    def __init__(self,t,q,n,p):
        "thread t to pop n integers from q"

    def run(self):
        "pops integers at some pace"
```

# Structure of Consumer Class

constructor and run

```
import threading
import random
import time

class Consumer(threading.Thread):
    """
    Pops integers from a queue.
    """
    def __init__(self,t,q,n,p):
        "thread t to pop n integers from q"

    def run(self):
        "pops integers at some pace"
```

# Structure of Consumer Class

constructor and run

```
import threading
import random
import time

class Consumer(threading.Thread):
    """
    Pops integers from a queue.
    """
    def __init__(self,t,q,n,p):
        "thread t to pop n integers from q"

    def run(self):
        "pops integers at some pace"
```

# Constructor of Consumer Class

```
def __init__(self,t,q,n,p):  
    "thread t to pop n integers from q"  
    threading.Thread.__init__(self,name=t)  
    self.queue = q  
    self.amount = n  
    self.pace = p
```

# Constructor of Consumer Class

```
def __init__(self,t,q,n,p):  
    "thread t to pop n integers from q"  
    threading.Thread.__init__(self,name=t)  
    self.queue = q  
    self.amount = n  
    self.pace = p
```

# Consuming Elements

the method run

```
def run(self):
    "pops integers at some pace"
    print "consumption starts..."
    for i in range(0,self.amount):
        r = random.randint(1,self.pace)
        print self.getName() + \
            " sleeps %d seconds" % r
        time.sleep(r)
    while True:
        try:
            i = self.queue.pop(0)
            print "popped %d from queue" % i
            break
        except IndexError:
            print "wait a second..."
            time.sleep(1)
    print "consumption terminated"
```

# Consuming Elements

the method run

```
def run(self):
    "pops integers at some pace"
    print "consumption starts..."
    for i in range(0,self.amount):
        r = random.randint(1,self.pace)
        print self.getName() + \
            " sleeps %d seconds" % r
        time.sleep(r)
    while True:
        try:
            i = self.queue.pop(0)
            print "popped %d from queue" % i
            break
        except IndexError:
            print "wait a second..."
            time.sleep(1)
    print "consumption terminated"
```

# Consuming Elements

the method run

```
def run(self):
    "pops integers at some pace"
    print "consumption starts..."
    for i in range(0,self.amount):
        r = random.randint(1,self.pace)
        print self.getName() + \
            " sleeps %d seconds" % r
        time.sleep(r)
    while True:
        try:
            i = self.queue.pop(0)
            print "popped %d from queue" % i
            break
        except IndexError:
            print "wait a second..."
            time.sleep(1)
    print "consumption terminated"
```

# Consuming Elements

the method run

```
def run(self):
    "pops integers at some pace"
    print "consumption starts..."
    for i in range(0,self.amount):
        r = random.randint(1,self.pace)
        print self.getName() + \
            " sleeps %d seconds" % r
        time.sleep(r)
    while True:
        try:
            i = self.queue.pop(0)
            print "popped %d from queue" % i
            break
        except IndexError:
            print "wait a second..."
            time.sleep(1)
    print "consumption terminated"
```

# the main program

prodcons.py

Code for the class `Producer` and `Consumer` in modules `classproducer` and `classconsumer` respectively.

```
from classproducer import *
from classconsumer import *

q = []          # queue is shared list
p = Producer("producer",q,1,3,4)
c = Consumer("consumer",q,3,1)

p.start()      # start threads
c.start()

p.join()       # wait for thread to finish
c.join()
```

# the main program

`prodcons.py`

Code for the class `Producer` and `Consumer` in modules `classproducer` and `classconsumer` respectively.

```
from classproducer import *
from classconsumer import *
```

```
q = []          # queue is shared list
p = Producer("producer",q,1,3,4)
c = Consumer("consumer",q,3,1)
```

```
p.start()      # start threads
c.start()
```

```
p.join()       # wait for thread to finish
c.join()
```

# the main program

`prodcons.py`

Code for the class `Producer` and `Consumer` in modules `classproducer` and `classconsumer` respectively.

```
from classproducer import *
from classconsumer import *

q = []          # queue is shared list
p = Producer("producer",q,1,3,4)
c = Consumer("consumer",q,3,1)

p.start()      # start threads
c.start()

p.join()       # wait for thread to finish
c.join()
```

## the main program

prodcons.py

Code for the class `Producer` and `Consumer` in modules `classproducer` and `classconsumer` respectively.

```
from classproducer import *
from classconsumer import *

q = []          # queue is shared list
p = Producer("producer",q,1,3,4)
c = Consumer("consumer",q,3,1)

p.start()      # start threads
c.start()

p.join()       # wait for thread to finish
c.join()
```

## the main program

prodcons.py

Code for the class `Producer` and `Consumer` in modules `classproducer` and `classconsumer` respectively.

```
from classproducer import *
from classconsumer import *

q = []          # queue is shared list
p = Producer("producer",q,1,3,4)
c = Consumer("consumer",q,3,1)

p.start()      # start threads
c.start()

p.join()       # wait for thread to finish
c.join()
```

# Summary + Assignments

## Concurrent Processes

processes and threads

life cycle of a thread

## Multithreading in Python

the thread module  
the Thread class

## Producer/Consumer Relation

Read chapter 13 in *Python Programming in Context*.

Assignments:

- 1 Implement the secret guessing with client/server network programming of lecture 24 using threads.
- 2 Modify the producer/consumer relationship into card dealing. The producer is the card dealer, the consumer stores the received cards in a hand.
- 3 When running a large simulation, e.g.: testing the distribution of a random number generator, it is useful to consider the evolution of the histogram. Design a multithreaded program where the producer generates random numbers that are then classified by the consumer.