Outline

1. Performance
   - Measurement

2. Concurrency
   - Multi-Thread
   - Multi-Process
   - Worker Pools

3. Distributed Computing
   - Shell Commands
   - Fabric

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
Time and Clock

- **time.time**
  - Returns the amount of time (in seconds) since the Epoch.
  - January 1, 1970 on UNIX and UNIX-based systems (e.g., Linux, OSX)
  - January 1, 1601 on Windows
  - Higher accuracy on UNIX machines

- **time.clock**
  - Behaves differently on UNIX and Windows machines.
  - Shows processor time on UNIX machines (ignores time sleeping)
  - Shows time since first call on Windows.
  - Higher accuracy on Windows machines.

*DO NOT DISTRIBUTE THIS MATERIAL*

*(Copyright Robert Rand, 2016)*
The `timeit` module times execution of bits of code.

Uses `time.clock` on Windows and `time.time` on everything else.

It avoids some common traps for timing code:
- Setup code is separated out and not timed
- Garbage collecting is turned off
- Repeated trials suppress measurement noise

Use `timeit` when you want to see which of 2 options is faster.
Using Timeit

import timeit

t = timeit.Timer(stmt=stmt_code, setup=setup_code)
t.timeit(number=num_trials)

- setup is executed once before any stmts
- stmt is executed num_trials times
- Returns time in seconds taken to execute
- The time does not include executing setup
- Copying the code to execute into a multi-line string could be useful
- A better idea is to import it:
  - setup = 'from __main__ import func_to_time'
Command Line and iPython Timeit

**Command Line**
- Use `python -m timeit 'command'`
- Include setup code as first argument with `-s`
- Chooses an appropriate number of iterations for you.
- Good for small snippets of mostly native code.

**iPython**
- Allows you to type `%timeit function` in the iPython REPL
- Has local scope: No need to import required functions.
Outline

1. Performance
   - Measurement

2. Concurrency
   - Multi-Thread
   - Multi-Process
   - Worker Pools

3. Distributed Computing
   - Shell Commands
   - Fabric

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
**Threading**

- **threading** is the built-in threading library
- **Create a thread:**

  ```python
  from threading import Thread
  args = (a1, a2, ...)
  kwargs = {k1:v1, k2:v2, ...}
  t = Thread(target=fun, args=args, kwargs=kwargs)
  t.start()
  ```

- **t.start():**
  - Creates a new thread in the current Python process
  - That thread then calls `fun(*args, **kwargs)`

---

**DO NOT DISTRIBUTE THIS MATERIAL**

(Permission Copyright Robert Rand, 2016)
Waiting on Threads

- When a thread is created it can execute in parallel
- Sometimes you need to be sure the Thread is done
- `t.join()` → Waits until thread t finishes
- If you create a bunch of threads to do a task
  - The task isn’t finished until all of the threads finish
  - You should not return a partial result to the caller
  - `.join()` on all the workers before finishing

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
CPython has a Global Interpreter Lock (GIL)
This means that only one thread can execute at a time
The exception is that threads release the GIL while doing I/O
The reason is to make the implementation of CPython simple
  ▶ Simple is better than complex
Take away:
  ▶ Multi-threaded Python code is not worth your time
  ▶ unless your doing a lot of I/O
Multi-processing

- **multiprocessing** is the built-in multiprocessing library

- Create a new process:

  ```python
  from multiprocessing import Process
  as = (a1, a2, ...)
  ks = {k1:v1, k2:v2, ...}
  p = Process(target=fun, as= args, ks=kwargs)
  p.start()
  
  p.start():
  - Creates a new Python process
  - That process then calls `fun(*args, **kwargs)`

- You should wait on processes with `p.join()`

(DO NOT DISTRIBUTE THIS MATERIAL)

(Permission Copyright Robert Rand, 2016)
Differences from Threads

**Threads (In Python)**
- Threads share memory
- Changing a variable in one thread can affect other threads
- Threads are **cheap** to make
- Threads basically need only a stack and Instruction Pointer

**Processes (In Python)**
- Processes do **not** share the same memory
- Processes are **expensive** to create
- A new process might copy all of the data of its parent
- Each process gets its own GIL
- Multiple processes actually run computations in parallel
Since Processes don’t share memory → need messages

```python
from multiprocessing import Queue

result_queue = Queue()
p = Process(target=func,
            args=(data, result_queue))
p.start()
ans = result_queue.get()
p.join()
```

If you try to `join` a process with a non-empty queue
  - The process won’t terminate
  - You may `deadlock`

---

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
Use a **pool** of worker processes instead of 1 process per task

- Creating a process is expensive
- Want to reuse the processes we already have

**concurrent.futures** provides pools of workers

```python
import concurrent.futures as cf
```

- `cf.ProcessPoolExecutor` creates workers using **multiprocessing**
- `cf.ThreadPoolExecutor` creates workers using **threading**

Map your workers to jobs

```python
cpus = os.cpu_count()
with cf.ProcessPoolExecutor(cpus) as ex:
    results = ex.map(function, [data1, ...])
```

*(Copyright Robert Rand, 2016)*
Concurrency is Complicated

- These are the basics for clearly separable tasks
- What to do if multiple threads want the same data?
  - Obstacles: Race Conditions, Starvation, Deadlock
  - Tools: Locks, Barriers, Semaphores, ...
- What if you want to run on multiple machines?
  - Distributed Computing

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
Outline

1. Performance
   - Measurement

2. Concurrency
   - Multi-Thread
   - Multi-Process
   - Worker Pools

3. Distributed Computing
   - Shell Commands
   - Fabric

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
The subprocess module allows execution of shell commands

- `subprocess.call('ls')`

The commands are run in a child process

Longer commands can be specified with a list of strings

- `call(['grep', '-ir', 'python', './'])`

The I/O of the subprocesses can be set with kwargs

- `call('ls', stdin=f_handle, stdout=DEVNULL)`
Alternatives to Call

- `subprocess.call`
  - Executes the command
  - Waits until it exits
  - Returns the exit code

- `subprocess.check_call`
  - Just like `call` but ...
  - If the exit code is not one (Abnormal exit) raise a `CalledProcessError`

- `subprocess.check_output`
  - Just like `check_call` but ...
  - returns the contents of stdout after the process finishes

- `subprocess.Popen`
  - Takes the same arguments as `call` (Except timeout)
  - Doesn’t wait for the process to finish
  - Have to check for output explicitly
  - More flexible for more complicated tasks
Piping

- Using `subprocess` you can pipe the output of one process to the input of another process.
- The easiest way: Just use `|` in a subprocess command.
  ```python
  subprocess.check_call(['ls', '|', 'wc', '-l'])
  ```
- `subprocess.PIPE` lets you do this with more control.
- Must use `subprocess.Popen` instead of `subprocess.call`:
  ```python
  subprocess.call waits for the process to finish
  If the pipe fills up then the process will deadlock
  ```
Installing Fabric

```
pip install fabric
```
Fabric is “a Python library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks.”

Fabric has two main features:

1. Easy use of the terminal from within a python program.
2. Allows multiple computers to easily communicate and coordinate tasks.
Invoking Fabric

$ fab func will call the func method in fabfile.py
  ▶ Optional arguments go after the colon.
  ▶ eg. $ fab hello:name=Bob will call hello with the argument name set to “Bob”.

If the file isn’t called fabfile.py use
$ fab -f [filename] [function_name]
Specifying Hosts

There are multiple ways of specifying the remote host(s).

1. `fab -H system1,system2,...` specifies the system(s) upon call.
2. `env.hosts = system1,system2,...` in the fabric file.
3. The decorators `@hosts(system1,system2,...)` can be applied to individual functions.
4. Otherwise, fabric will ask for the system upon execution.
Commands

- **local(command)** runs a command on the local machine.
  - Set `capture = True` to return the output instead of sending it to `stdout`.

- **run(command)** runs a command on the remote machine.
  - Set `stdin = x` and `stdin = y` to pass `stdin` and `sterr` to the variables `x` and `y`.  

**DO NOT DISTRIBUT THIS MATERIAL**

*(Copyright Robert Rand, 2016)*
Get and Put

- Both `get` and `put` take in a `local_path` and `remote_path`, relative to the current local and remote directories. (Absolute paths are also permitted.)
- `put` transfers a file from the local machine to the remote machine.
- `get` does the reverse.
- Use `cd(dir)` to run all commands from a given directory.
For more on Fabric, check out:

http://docs.fabfile.org/en/1.11/tutorial.html
CIS192 Python Programming
Graphical User Interfaces

Robert Rand
University of Pennsylvania
April 20, 2016

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
Outline

1 Graphical User Interface
   - Tkinter
   - Other Graphics Modules

2 Text User Interface
   - curses
   - Other Text Interface Modules

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
Tkinter

- The module **Tkinter** is a wrapper of the graphics library **Tcl/Tk**
- **Why choose Tkinter over other graphics modules**
  - It’s bundled with Python so you don’t need to install anything
  - It’s fast
  - Guido van Rossum helped write the Python interface
- The docs for **Tkinter** aren’t that good
  - The docs for Tk/Tcl are much better
  - Tk/Tcl functions translate well to Tkinter
  - It’s helpful to learn the basic syntax of Tk/Tcl
- **Tk/Tcl syntax → Python:**
  - `class .var_name -key1 val1 -key2 val2 → var_name = class(key1=val1, key2=val2)`
  - `.var_name method -key val → var_name.method(key=val)`

(Copyright Robert Rand, 2016)
from Tkinter import Frame

class SomeApp(Frame):
    def __init__(self, master=None):
        tk.Frame.__init__(self, master)

def main():
    root = tk.Tk()
    app = SomeApp(master=root)
    app.mainloop()

if __name__ == '__main__':
    main()

(Copyright Robert Rand, 2016)
Images

- By default, Tkinter only supports bitmap, gif, and ppm/pgm images
- More images are supported with Pillow
- Pillow is a fork of Python Imaging Library `pip install pillow`
- `from PIL import Image, ImageTk`
- Create a PIL image with `img = Image.open('path_to_img')`
- Make a Tk image with `tk_img = ImageTk.PhotoImage(im)`
- Set it as an attribute in Tkinter with `b['image'] = tk_img`
Tkinter has a bunch of widgets

- Button, Label, Listbox, Radiobutton

Create a widget with \( b = \text{Button}(\text{parent}) \)

- \( \text{parent} \) is the containing widget

Options can be accessed and set dictionary style

- \( b[\text{'}\text{text}\text{'}] = \text{'Press Me'} \)
- equivalently: \( b = \text{Button}(\text{parent, text='Press Me'}) \)
Placing Widgets

- Just creating a widget will not display it
- The widget must be told where to go in the parent widget
- **Grid placement**: `my_widget.grid(row=r, column=c)`
  - Best to specify the grid layout beforehand.
- **Absolute placement**: `my_widget.place(x=c1, y = c2)`
  - May conflict with other objects.
  - Will ignore window resizing.
- **Packing**: `my_widget.pack(side = SIDE)`
  - Will be placed relative to other widgets.
  - Can specify `padx` and `pady` padding.
Handlers

- A Widget can have a registered Event Handler
- The handler is a function that gets called when the widget is used
- Register a handler for a widget:
  \[
  b[\text{'command'}] = \text{some\_function}
  \]
- Handlers do not automatically say what widget was used
  - Use a lambda to partially apply arguments
    - \[
    b[\text{'command'}] = \text{lambda } w: \text{button\_handler}(w)
    \]
  - Or use \text{from functools import partial}
    - \[
    b[\text{'command'}] = \text{partial}(\text{button\_handler}, w)
    \]
Other GUIs

- **WxPython**
  - Similar to Tkinter in that it wraps an existing library
  - Wraps the C++ wxWidgets
  - A little bit more user-friendly

- **PyQT and PySide**
  - Python bindings for the Qt cross-platform application and UI framework
  - PyQT is commercial software, PySide is open source

- **PyGTK**
  - Implements GTK+ (originally the “GIMP Toolkit”).
  - Requires a separate GTK+ install.
Outline

1 Graphical User Interface
   • Tkinter
   • Other Graphics Modules

2 Text User Interface
   • curses
   • Other Text Interface Modules

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
curses is a wrapper around the ncurses library

ncurses is the standard for terminal graphics
- Is terminal independent (XTerm, Command Prompt, ...)
- Treats screen as a grid of characters
- Pretty low-level

An curses program runs in your current terminal
- Not a new window
- Debugging with print statements can cause weird behavior
- Changes made in the program can persist after termination
- curses.wrapper ensures that clean-up happens on termination

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
import curses

class SomeApp(object):
    def __init__(self, stdscr):
        self.stdscr = stdscr

    def run(self):
        while True:
            key = self.stdscr.getch()

def main(stdscr):
    app = SomeApp(stdscr)
    app.run()

if __name__ == '__main__':
    curses.wrapper(main)
Wrapper Explained

- **wrapper**(main) **executes** **main**(stdscr) **in a** **try/except**
- **stdscr** **is an initialized** **curses** WindowObject
- The initialization includes:
  - cbreak: Buffering is turned off (But Ctrl-C still works)
  - no echo: Typed characters are not displayed on screen
  - colors: If the terminal supports colors they are initialized
- before exiting the settings are reset

DO NOT DISTRIBUTE THIS MATERIAL

(Permission Copyright Robert Rand, 2016)
Writting Strings

- A WindowObject is a uniform grid of characters
- Given a WindowObject \( w \)
  - \( w.addstr(row, \text{column}, \text{some_string}) \) will write \text{some_string} to the window starting at \((row, \text{column})\)
- Overwriting a section of a window will only replace those characters
  - Use \( w.clear() \) to clear the entire window
- For the effects of a write or clear to take effect
  - \( w.refresh() \) repaints the window
  - \( w.noutrefresh() \) marks the window for update
  - \( w.doupdate() \) actually repaints the screen
  - \( w.refresh() \) marks the current window and repaints all marked windows

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
Creating Windows

- A **window** is basically a name for a rectangle of the screen
- `curses.newwin(height, width, r, c)` creates a window starting at row=r and column=c
- Windows allow parts of the screen to be refreshed separately
- Windows give a new coordinate system with (0, 0) in the top-left
- A **panel** is a window with depth
- You can overlap panels without overwriting other panels data
Setting Attributes

- When writing a string you can specify Attributes
- Background/Foreground color pairs:
  - red = curses.COLOR_RED
  - black = curses.COLOR_BLACK
  - curses.init_pair(1, red, black)
  - w.addstr(0, 0, 'some_text', curses.color_pair(1))
- Bolding and Highlighting:
  - w.addstr(0, 0, 'other_text', curses.A_BOLD)
  - w.addstr(0, 0, 'other_text', curses.A_BLINK)
- Attributes are not guaranteed to mix well but multiple can be specified
- Attributes can be applied to entire windows
  - w.bkgd(0, curses.A_STANDOUT)

(Copyright Robert Rand, 2016)
key = w.getch() waits for a key to be pressed
The return value is an integer representing the character
Compare against constants to detect special keys
  if key == curses.KEY_RIGHT:

w.getkey() will return a string instead of an integer
Control Logic

- `curses` is very low-level
- Minimal abstraction (Rectangles of characters)
- No notion of event handlers
  - All key-presses and mouse clicks must be explicitly directed
- You are in charge of all state

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
Example: cmus

- a small fast music player written in curses
Urwid

- Urwid is a text widget library
- Has more abstraction (Widgets instead of blocks of text)
- If your UI is that complex just use a GUI
CIS192 Python Programming
Probability and Simulations

Robert Rand
University of Pennsylvania

March 02, 2016
1 Probability and Monte Carlo Methods
   - The Two Child Problems
   - Basic Definitions
   - The Law of Large Numbers
   - Monte Python
   - Conditional Probability
   - Bayes Theorem

2 PAC Learning

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
The Two Child Problem

Suppose I have two children.

I announce that I have a son.

What is the probability that both children are boys?
### Considering the Possibilities

<table>
<thead>
<tr>
<th>First child</th>
<th>Second child</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boy</td>
<td>Boy</td>
</tr>
<tr>
<td>Boy</td>
<td>Girl</td>
</tr>
<tr>
<td>Girl</td>
<td>Boy</td>
</tr>
<tr>
<td>Girl</td>
<td>Girl</td>
</tr>
</tbody>
</table>

**DO NOT DISTRIBUTE THIS MATERIAL**

(Copyright Robert Rand, 2016)
I announce that

“I have a son who was born on a Tuesday.”

Now what is the probability that both children are boys?
A Handy Visualization

The Tuesday Child Problem, visually demonstrated

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
Definition: A *Random Variable* is a function from a set of possible outcomes to associated (generally numeric) values.

- For a coin toss we’ve been mapping [lands heads] to 1 and [lands tails] to 0.
- The values associated with a die roll are the numbers 1 through 6.

- Commonly represented by capital $X$.
- We will model random variables with (random) Python functions.
Each random variable has an associated *Probability Distribution* that maps its values to *probabilities*.

- Every probability must be in the interval $[0, 1]$.
- The sum of the probabilities must equal 1.

We say $Pr(X = x) = p$ do denote the probability of an event.

- eg. for a fair die $D$, $Pr(D = 3) = \frac{1}{6}$
$E(X) = \sum_x Pr(x) \cdot x$

- Where $C$ represents a coin toss as above, $E(C) = 0.5 \cdot 1 + 0.5 \cdot 0 = 0.5$
- For a die $D$, $E(D) = \frac{1}{6} \cdot 1 + \cdots + \frac{1}{6} \cdot 6 = 3.5$
The Law of Large Numbers

The Weak Law of Large Numbers states that as the number of trials approaches infinity, the frequency of a given outcome approaches its true probability.

- i.e. If we toss a lot of coins, we will get heads close to half the time.
- Which brings us back to Python.
You’re on a gameshow.

There are three doors in front of you, two have goats behind them and one has a car. After you pick a door, Monty Hall (the host) opens another door, revealing a goat. He then gives you the option of switching.

Should you switch?
```python
def monty(switch = False):
    car = randint(1,3)
pick = randint(1,3)
open = choice(list({1, 2, 3} - {car, pick}))
if switch:
    pick = choice(list({1, 2, 3} - {pick, open}))
if pick == car:
    return "Car"
else:
    return "Goat"
```

DO NOT DISTRIBUTE THIS MATERIAL

(Permission Robert Rand, 2016)
Incentives

...and my yard has so much grass, and I'll teach you tricks, and...

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
def monty(switch = False, evil = False):
car = randint(1,3)
pick = randint(1,3)
if evil and not pick == car:
    return "Goat"
open = choice(list({1, 2, 3} - {car, pick}))
if switch:
    pick = choice(list({1, 2, 3} - {pick, open}))
if pick == car:
    return "Car"
else:
    return "Goat"
Sometimes we’re interested in the probability of two events occurring together. For example, we might be interested in the probability that Monty is evil AND we win a car.

We represent this as $Pr(X, Y)$ or $Pr(X \cap Y)$

i.e. $Pr(result = Car, Monty = Evil)$
Conditional Probability

Alternatively we’re interested in the probability of some event happening once we know that another event has occurred. For example, we might be interested in the probability that we win a car GIVEN the knowledge that Monty is evil.

We represent this as $Pr(X \mid Y)$
i.e. $Pr(result = Car \mid Monty = Evil)$

$$Pr(X \mid Y) = \frac{Pr(X \cap Y)}{Pr(Y)}$$

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
Independence

If the event $Y$ doesn’t impact the probability of $X$, that is:

$$Pr(X \mid Y) = Pr(X)$$

we say that $X$ and $Y$ are independent.
Bayes Nets

Switch -> Prize -> Open -> Goatee
Evil -> Prize

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
Bayes’ Theorem

\[ Pr(X \mid Y) = \frac{Pr(Y \mid X)P(X)}{P(Y)} \]

This follows directly from the definition of conditional probability:

\[ Pr(X \mid Y)P(Y) = Pr(X \cap Y) = P(Y \mid X)P(X) \]
1. Probability and Monte Carlo Methods
   - The Two Child Problems
   - Basic Definitions
   - The Law of Large Numbers
   - Monte Python
   - Conditional Probability
   - Bayes Theorem

2. PAC Learning

DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
We say that a concept \( c \) is **Probably Approximately Correct (PAC) Learnable** if we have an time algorithm that:

1. Takes an arbitrary error bound \( \epsilon \) and failure probability \( \delta \),
2. Takes in an arbitrary distribution of labeled items \( D \),
3. Returns an approximation of the concept \( c_a \) that (with probability greater than \( 1 - \delta \)) correctly classifies \( (1 - \epsilon) \)-fraction of the points, and
4. Runs in polynomial time in \( \epsilon \) and \( \delta \).

**DO NOT DISTRIBUTE THIS MATERIAL**

*(Copyright Robert Rand, 2016)*
The concept $c$ might be “people who are the right height and weight to become US Air Force pilots”.

Say our distribution is a randomly selected group of adults. We send them all to the airforce academy and record their height, weight and whether they passed.

We will then estimate the minimum weight as the weight of the lightest person who passed the test, and likewise for max weight, and min height and weight.

We won’t go through all the math here, but it should only take $\approx 1/\varepsilon$ people to come within $\varepsilon$ of each parameter, so $\approx 4/\varepsilon$ to approximate all four borders. We then scale by a function of $\delta$.
DO NOT DISTRIBUTE THIS MATERIAL

(Copyright Robert Rand, 2016)
PAC Caveat

We’re only guaranteeing high accuracy classification of samples drawn from our distribution.

Hence, if we’re testing toddlers, we won’t have any who pass the test, and our approximate test will always say “No”.

Fortunately, that’s the correct response to 100% of toddlers who wish to enroll in the Air Force.
Random Functions

- `random()` returns a float between 0 and 1 uniformly at random
- `randint(a, b)` returns a random integer between $a$ and $b$ inclusive
- `choice(li)` samples a random element from `li`
- `sample(li)` returns a randomly selected subset of `li`
- `shuffle(li)` randomly permutes the list `li`
- `uniform(a, b)` returns a float uniformly distributed between $a$ and $b$
- `gauss(mean, stddev)` returns a float from a gaussian (or normal) distribution with the given mean and standard deviation.

DO NOT DISTRIBUTE THIS MATERIAL

(Permission granted to Robert Rand, University of Pennsylvania for classroom presentation and educational use)

(Copyight Robert Rand, 2016)