How your computer works

CS106AP Lecture 26
Roadmap

Programming Basics

The Console

Images

Data structures

Midterm

Graphics

Object-Oriented Programming

Everyday Python

Tuples

List Comp.

Lambdas

Computers

Internet

Life after...

Exam review

In-class OH
Today’s questions

How does your computer work?
Today’s topics

1. Review
2. GlImages (Assignment 6)
3. 3 parts of your computer
4. Binary
5. What’s next?
Python techniques for data manipulation

- Tuples
- Comprehensions
- Lambdas
Python techniques for data manipulation

- **Tuples**
  - Bundle small amounts of related information together
  - Number of elements is known and limited
  - *Immutable* data type

- **Comprehensions**

- **Lambdas**
Python techniques for data manipulation

- Tuples

- Comprehensions
  - Iterate over an existing data structure
  - Create a **new list or dictionary** more efficiently
  - Can filter elements using if statements

- Lambdas
Python techniques for data manipulation

- Tuples

- Comprehensions

- Lambdas
  - More memory efficient, one-line functions without names
  - Best used with min(), max(), and sorted() for custom sorting of data
  - Avoid writing a lambda function that only calls an existing function
Lambda functions
Lambda functions

\[
\text{lambda } n: n * 2
\]

\[
\text{lambda } x, y: x ** y
\]

\[
\text{lambda } \text{tup}: \text{tup}[0]
\]

**Definition**

Lambda
A one-line, unnamed function

**Note:**
no def, no return
Lambdas vs. Regular Functions

lambda x: x * 2

def double(x):
    return x * 2
Lambdas vs. Regular Functions

\[
\text{lambda } x: x \times 2
\]

\[
\text{def double(x):}
\]
\[
\quad \text{return } x \times 2
\]
Lambdas vs. Regular Functions

lambda x: x * 2

def double(x):
    return x * 2
Lambdas vs. Regular Functions

```
lambda x: x * 2
```

def double(x):
    return x * 2
Lambdas vs. Regular Functions

**Lambdas**

```python
def double(x):
    return x * 2
```

**Regular Functions**

```python
lambda x: x * 2
```

This expression is automatically returned.

We need `return` in order to return!
Why lambdas?

- Great for when you need a tiny function
- Use less memory than regular functions in Python
- Powerful in the context of custom sort and min/max
Lambdas and `sorted()`

- `sorted(iterable, key, reverse)`
  - `key` is where you can pass in a lambda
  - The `key` function transforms each element before sorting them.
    - It outputs the value to use for comparison when sorting.
Lambdas and `sorted()`

```python
>>> fruit = [('mango', 3), ('apple', 6), ('lychee', 1), ('apricot', 10)]

>>> sorted(fruit)
[('apple', 6), ('apricot', 10), ('lychee', 1), ('mango', 3)]

# get the second value from the tuple and sort on it

>>> sorted(fruit, key=lambda elem: elem[1])
[('lychee', 1), ('mango', 3), ('apple', 6), ('apricot', 10)]
```
Lambdas and \texttt{min()}/\texttt{max()}

- \texttt{min(iterable, key)}
  - If you just care about the smallest or largest element, \texttt{min()} and \texttt{max()} are faster and less costly than sorting a list.
  - The \texttt{key} function transforms each element before comparing them to find the min/max.

\texttt{key is an optional argument}
Lambdas and `min()`/`max()`

```python
g>>> fruit = [('mango', 3), ('apple', 6), ('lychee', 1), ('apricot', 10)]

g>>> min(fruit)
('apple', 6)

g>>> max(fruit)
('mango', 3)

g>>> min(fruit, key=lambda elem: elem[1])
('lychee', 1)

g>>> max(fruit, key=lambda elem: elem[1])
('apricot', 10)
```
New function: `sum()`

- `sum(iterable)`
- Returns the sum of the elements contained in a list, dict, or tuple.
matplotlib
matplotlib

- A library for creating plots
  - Especially useful inside Jupyter notebooks

- To install:

  ```
  $ python3 -m pip install matplotlib
  ```
Using matplotlib

import matplotlib.pyplot as plt

# x = list of x vals, y = list of y vals
plt.plot(x, y)  # line or scatter plot
plt.scatter(x, y)  # scatter plot
plt.title(text)  # adds a title
plt.show()  # display
Using matplotlib

```python
import matplotlib.pyplot as plt

# x = list of x vals, y = list of y vals
plt.plot(x, y)  # line or scatter plot
plt.scatter(x, y)  # scatter plot
plt.title(text)  # adds a title
plt.show()  # display
```
import matplotlib.pyplot as plt

# x = list of x vals, y = list of y vals
plt.plot(x, y)  # line or scatter plot
plt.scatter(x, y)  # scatter plot
plt.title(text)  # adds a title
plt.show()  # display

Make sure the list indices correspond to the same plot points!
Assignment 6 is out!
Assignment 6 has two parts

● A Jupyter notebook to practice tuples, comprehensions, and lambdas

● An open-ended graphics program
Assignment 6 has two parts

- A Jupyter notebook to practice tuples, comprehensions, and lambdas
- An open-ended graphics program

We've added new campy functionality to help!
Glimages

- Like SimpleImages but as GObjects!
GlImages

- Like SimpleImages but as GObjects!

```python
image.height
image.width
image.get_pixel(x, y)
image.set_pixel(x, y, pixel)
```

SimpleImage functions!
Glimages

- Like SimpleImages but as GObjects!

```python
image.x
image.y
image.move(dx, dy)
```
GImages

● Like SimpleImages but as GObjects!

● A few differences...
  ○ Declaration
    
    ```python
    image = GImage.from_file(filename_string)
    ```

  ○ No image resizing or creation of blank images

  ○ Graphics **Pixel** objects are not associated with the image
GImages

- Like SimpleImages but as GObjects!

- A few differences...
  - Declaration
    
    ```python
    image = GImage.from_file(filename_string)
    ```
  - No image resizing or creation of blank images
  - Graphics **Pixel** objects are not associated with the image

What does this mean?
Pixel objects in a GImage

- **Pixel** objects don’t have \(x, y\) attributes.

- Editing the pixel won’t update the image.

- This means that you should not for each loop over the image or you won’t have access to the pixel’s coordinates!
How to loop over a GImage

You must call .set_pixel() to update the image after editing the pixel!
How to loop over a GImage

You must call `.set_pixel()` to update the image after editing the pixel!

```python
for y in range(image.height):
    for x in range(image.width):
        # Use pixel = image.get_pixel(x, y) to get pixel
        # Edit pixel.red, pixel.green, pixel.blue
        # Use image.set_pixel(x, y, pixel) to update pixel
        # within the image
```
How does your computer work?
A restaurant analogy

Your computer has three main parts!
1. Storage

This is the file system on your computer – where all of your persistent data (e.g. photos, code files, word docs etc.) live
1. Storage

2. Central processing unit (CPU)

This is your computer’s brains and brawn! (what does all the work of understanding and “running” your code)
This is your computer’s temporary memory – where it stores information for the current task(s) at hand (e.g. variables, the code its currently running, open applications on your computer, etc.)
1. Storage
2. Central processing unit (CPU)
3. Random Access Memory (RAM)

Users (us!)
What does it look like to run my code?
Running a program

- Your code file is stored in storage (persistent memory).
Running a program

- Your code file is stored in storage (persistent memory).

- The CPU runs your program and is powered by one or multiple cores.
Running a program

- Your code file is stored in storage (persistent memory).
- The CPU runs your program and is powered by one or multiple cores.
- When you “run” programs (or applications), each program gets its own space in RAM.
  - For your Python scripts, the code file’s lines of code get moved to RAM, and any created Python objects also get their own space in RAM.
Running a program

- Your code file is stored in storage (persistent memory).
- The CPU runs your program and is powered by one or multiple cores.
- When you “run” programs (or applications), each program gets its own space in RAM.
- When the program exits, the space it took in RAM gets reclaimed.
  - Who keeps all this organized in RAM? Your operating system (OS)!
The operating system

- It’s the first program that gets run when you turn on your computer.
The operating system

- It’s the first program that gets run when you turn on your computer.

- It stops and starts other programs and manages the space they take up in RAM.
The operating system

- It’s the first program that gets run when you turn on your computer.
- It stops and starts other programs and manages the space they take up in RAM.
- It’s what we’re talking to when we type commands into Terminal/the command line!
The operating system

- It’s the first program that gets run when you turn on your computer.
- It stops and starts other programs and manages the space they take up in RAM.
- It’s what we’re talking to when we type commands into Terminal/the command line!
CPU and RAM: a demo
(follow along at your own risk)
Why do people always write 0s and 1s to represent "code"?
What is RAM exactly?

- When you store information in Python, it becomes a Python object
  - Objects come in different sizes and types
Computers as switches

- Your computer’s hardware is made up of many tiny switches!
  - You can also think of switches as closed or open circuits.

On = 1

Off = 0
Computers as switches

- Your computer’s hardware is made up of many tiny switches!
- Each switch can have one of two values: on (1) or off (0)
Computers as switches

- Your computer’s hardware is made up of many tiny switches!

- Each switch can have one of two values: on (1) or off (0)

**Definitions**

- **bit** - a single 0 or 1 digit
- **byte** - 8 bits
But how do we actually represent numbers?
Normal number (‘base 10’)

25,306

(2 \times 10^4) + (5 \times 10^3) + (3 \times 10^2) + (0 \times 10^1) + (6 \times 10^0)
Normal number ("base 10")

25,306

\[
\begin{array}{ccccc}
10^4 & 10^3 & 10^2 & 10^1 & 10^0 \\
2 & 5 & 3 & 0 & 6 \\
\end{array}
\]

20,000 + 5,000 + 300 + 0 + 6
Normal number ("base 10")

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 9 \\
10^4 & 10^3 & 10^2 & 10^1 & 10^0 \\
\end{array}
\]

\[(9 \times 10^0)\]
Binary numbers ("base 2")

<table>
<thead>
<tr>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

9?
Binary numbers ("base 2")

```
   0  0  0  0  0  0
   16  8  4  2  1
```

9?
Binary numbers ("base 2")

9?

\[
\begin{array}{cccccc}
0 & 1 & 0 & 0 & 0 & 0 \\
16 & 8 & 4 & 2 & 1 & \text{9?} \\
\end{array}
\]

\[(1 \times 2^3)\]
Binary numbers ("base 2")

(1 \times 2^3) + (1 \times 2^0) = 8 + 1 = 9
Counting binary patterns

One switch (one bit) gives us only two possible patterns:

0 or 1

If we have two switches, how many different patterns are there?
Counting binary patterns

One switch (one bit) gives us only two possible patterns:

0 or 1

If we have two switches, how many different patterns are there?

00, 01, 10, 11 → 4 patterns
## Counting binary patterns

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Number of patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

- 1, 0
- 00, 01, 10, 11
Counting binary patterns

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Number of patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

1, 0

00, 01, 10, 11

000, 001, 010, 011, 100, 101, 110, 111
Counting binary patterns

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Number of patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

1, 0

00, 01, 10, 11

000, 001, 010, 011, 100, 101, 110, 111

(i.e. 4 x 2 patterns)
Counting binary patterns

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Number of patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
</tbody>
</table>

1, 0
00, 01, 10, 11
000, 001, 010, 011, 100, 101, 110, 111
8 x 2 patterns
Counting binary patterns

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Number of patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2^1$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3$</td>
</tr>
<tr>
<td>4</td>
<td>$2^4$</td>
</tr>
</tbody>
</table>
## Counting binary patterns

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Number of patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2^1$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3$</td>
</tr>
<tr>
<td>4</td>
<td>$2^4$</td>
</tr>
</tbody>
</table>

8 bits = 1 byte

$2^8 = ?$
Counting binary patterns

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Number of patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2^1$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3$</td>
</tr>
<tr>
<td>4</td>
<td>$2^4$</td>
</tr>
</tbody>
</table>

8 bits = 1 byte
$2^8 = 256$
Counting binary patterns

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Number of patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2^1$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3$</td>
</tr>
<tr>
<td>4</td>
<td>$2^4$</td>
</tr>
</tbody>
</table>

8 bits = 1 byte
$2^8 = 256$

The number of possible RGB values!
## Counting binary patterns

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Number of patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2^1$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3$</td>
</tr>
<tr>
<td>4</td>
<td>$2^4$</td>
</tr>
</tbody>
</table>

8 bits = 1 byte  
$2^8 = 256$

Each color channel takes up 1 byte!
Counting binary patterns

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Number of patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2^1$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3$</td>
</tr>
<tr>
<td>4</td>
<td>$2^4$</td>
</tr>
</tbody>
</table>

8 bits = 1 byte
$2^8 = 256$

Each pixel has a red byte, a green byte, and a blue byte!
What’s next?