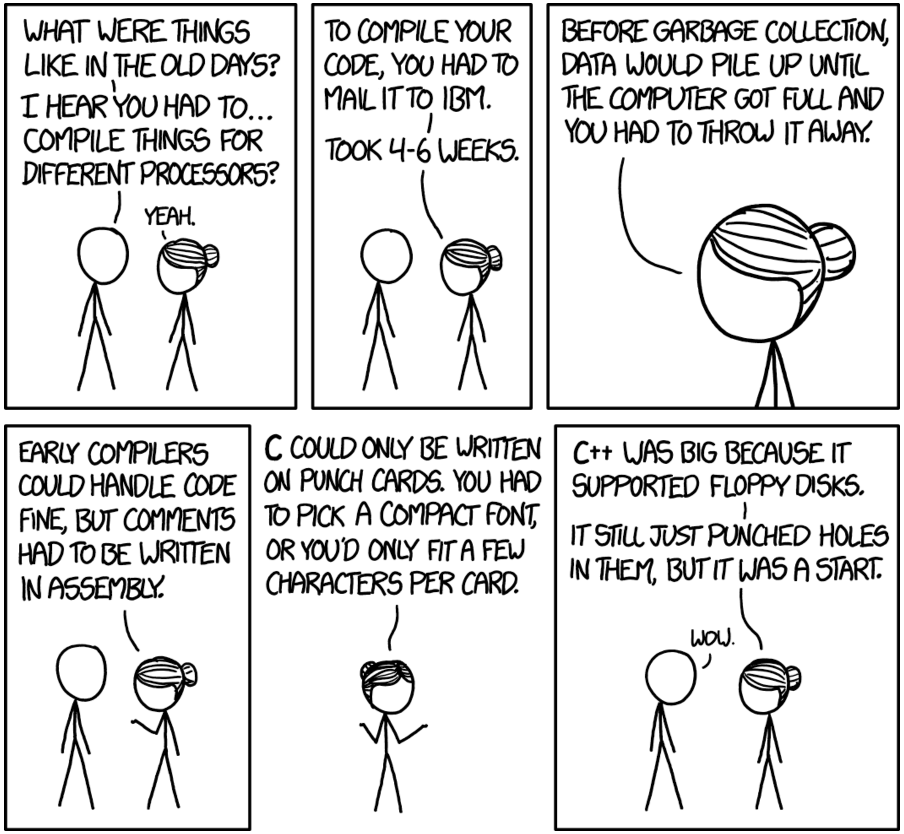
ENGSCI 233 Lecture 12.1

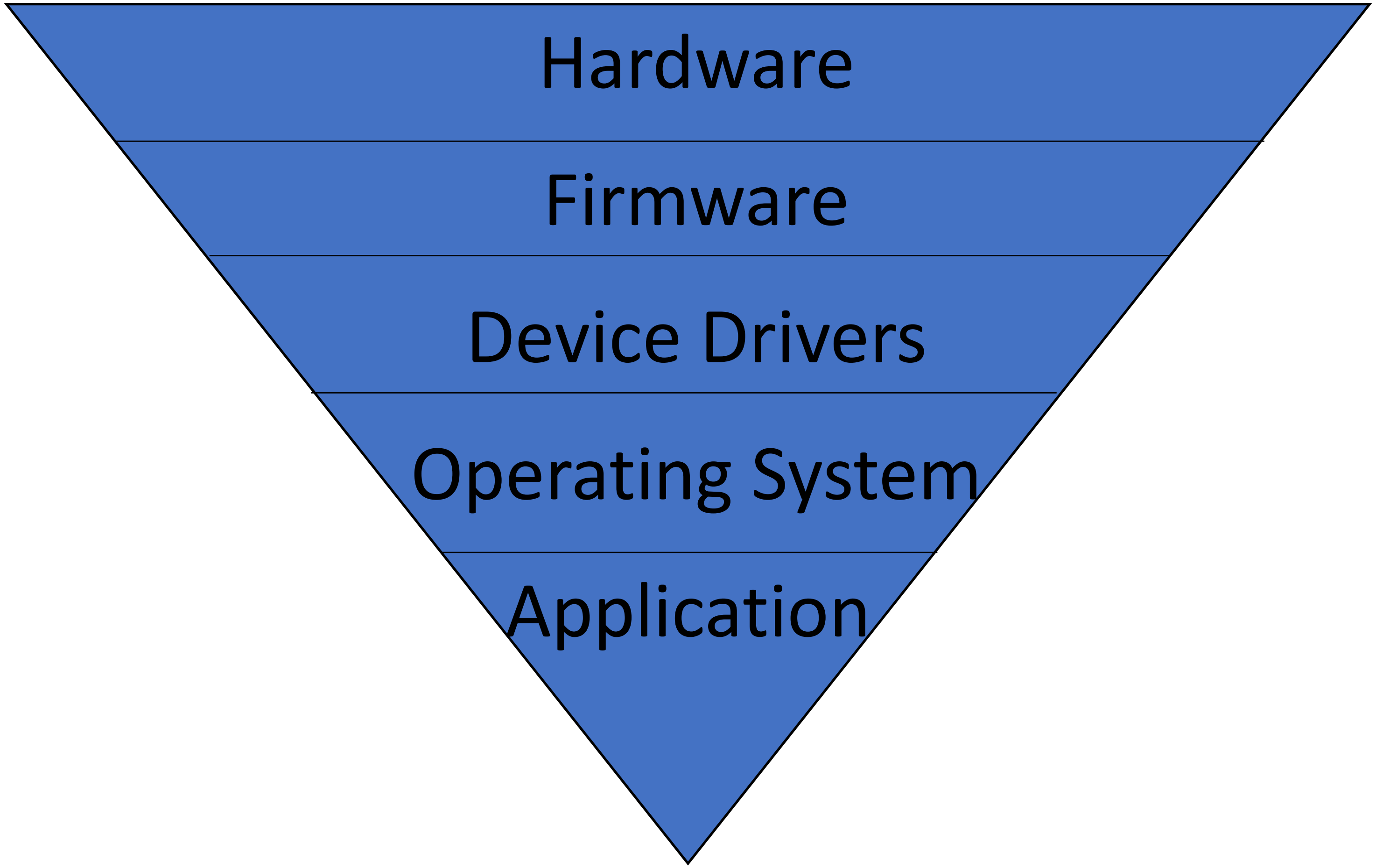
Software Systems


WHAT WERE THINGS LIKE IN THE OLD DAYS?
I HEAR YOU HAD TO... COMPILE THINGS FOR DIFFERENT PROCESSORS?
YEAH.
TO COMPILE YOUR CODE, YOU HAD TO MAIL IT TO IBM.
TOOK 4-6 WEEKS.
BEFORE GARBAGE COLLECTION, DATA WOULD PILE UP UNTIL THE COMPUTER GOT FULL AND YOU HAD TO THROW IT AWAY.
EARLY COMPILERS COULD HANDLE CODE FINE, BUT COMMENTS HAD TO BE WRITTEN IN ASSEMBLY.
C COULD ONLY BE WRITTEN ON PUNCH CARDS. YOU HAD TO PICK A COMPACT FONT, OR YOU'D ONLY FIT A FEW CHARACTERS PER CARD.
C++ WAS BIG BECAUSE IT SUPPORTED FLOPPY DISKS.
IT STILL JUST PUNCHED HOLES IN THEM, BUT IT WAS A START.
WOW.

Today’s learning objectives:

- Understand the software components of a computer system.
- Describe the main forms of memory management used in computer systems.
- Understand how APIs are used to access device drivers, operating systems, and other people’s applications.

Hardware is only one part.



Hardware

Firmware

Device Drivers

Operating System

Application

Hardware is only one part.

- Firmware – software that runs on dedicated, low-performance processors
- Device drivers – software that provides a common interface to firmware/hardware
- Operating system – software that manages the operation of other software in a computer system
- Application – the software that actually accomplishes your task

System design is flexible.

- Not all computer systems need all the levels.
- Simple *embedded systems* have only firmware and hardware
- Computer systems can be made up of multiple other, smaller computer systems
    - E.g. “the cloud”
- One set of hardware can run several computer systems at once
    - *Virtualization*

How can software use hardware?

- Focus today on memory
  - RAM
- How much do we need?
- What if it changes?
- How do we use it?

Memory is used 3 ways.

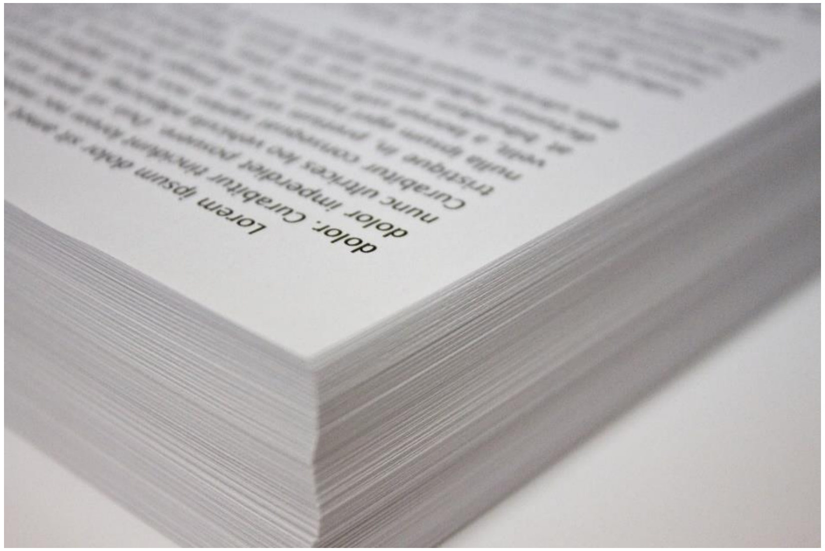
- **Static allocation**
- Stack
- Heap

Static allocations are frozen.

- Memory address determined at compile time
- Always needs space at all times
- Cannot vary size of variables
- Has to be planned in detail while writing code
- Typically for global variables in your program
- In Python:
    - No static allocations are possible.

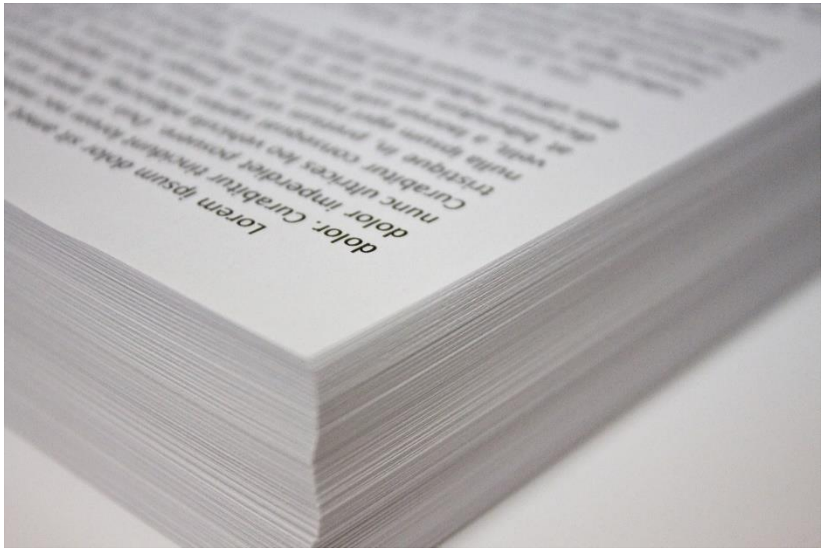
Memory is used 3 ways.

- Static allocation
- **Stack**
- Heap



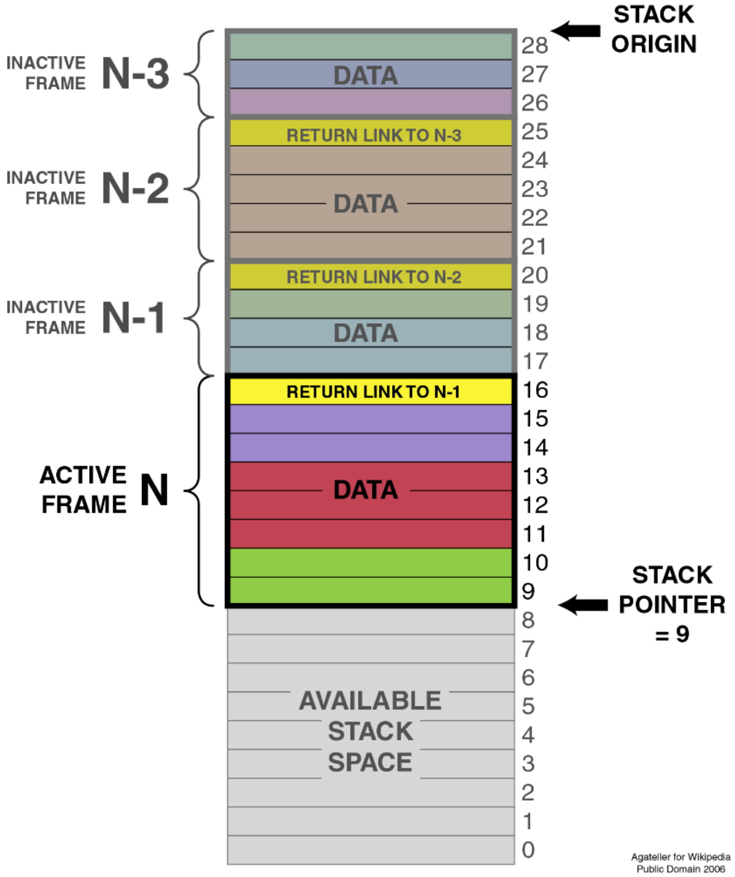
The stack makes functions work.

- Memory address determined when function begins executing
- Last-in-first-out structure
- Managed in part by processor hardware



- What does this look like?

How does the stack work?



Agateller for Wikipedia
Public Domain 2006

- Calling function puts memory address on stack to start new frame
- Local variables allocated at start of function in this frame
- When function ends, active frame becomes free

Why should I care about the stack?

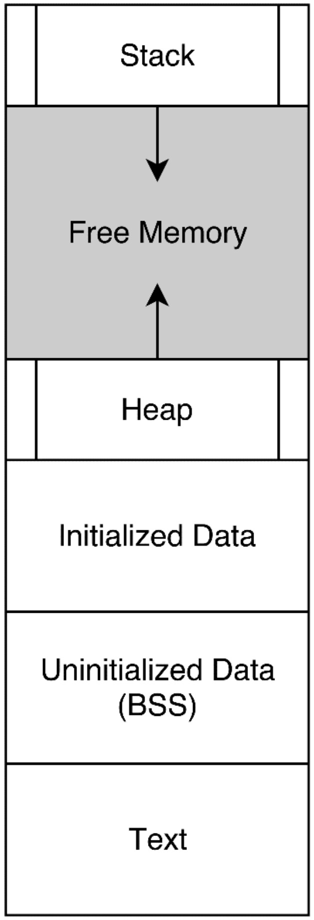
- Functions that call themselves can use a lot of stack memory!
- Some hardware has a limited stack depth
- Stack overflow = crash!

Memory is used 3 ways.

- Static allocation
- Stack
- **Heap**

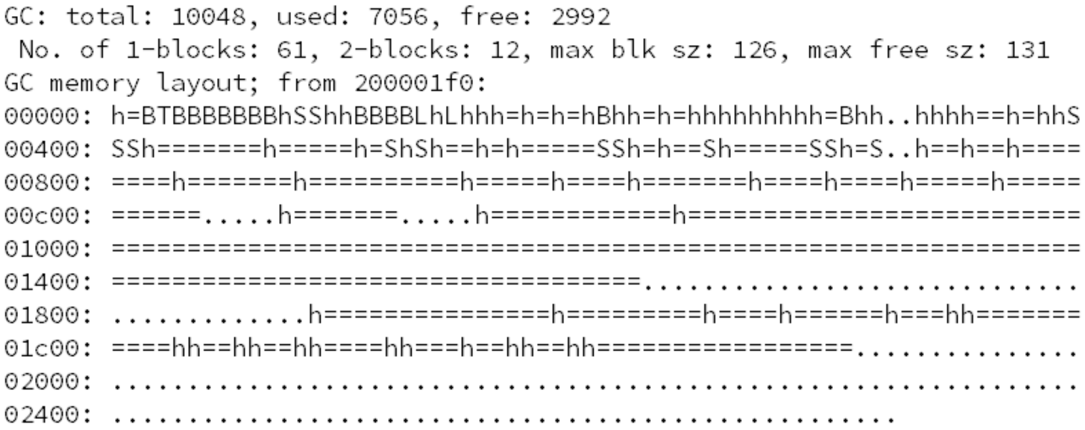
The heap is dynamically allocated.

- Variables can be created or destroyed
- Arrays can change size
- Complex code objects can exist



How does the heap stay organized?

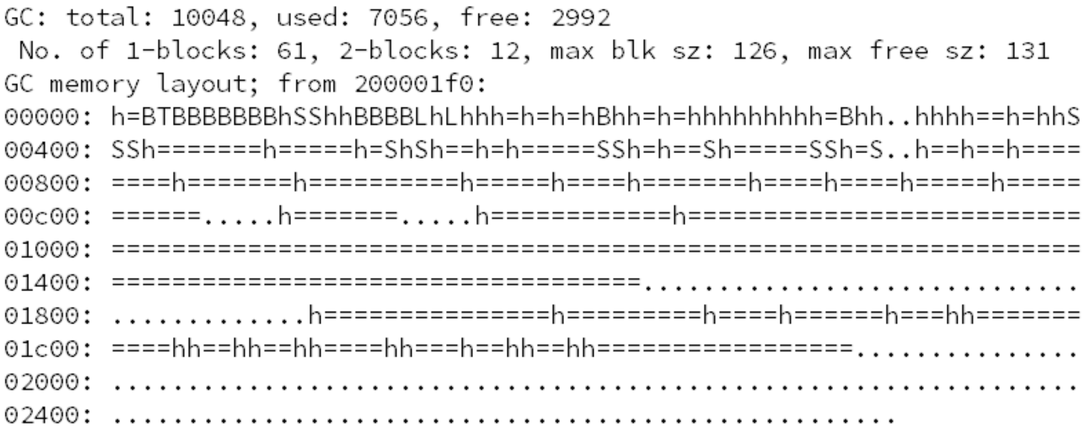
- Allocated in blocks
- Needs a memory manager for allocation
- How to de-allocate?
  - Manually (C)
  - Garbage collection (Python)



```
GC: total: 10048, used: 7056, free: 2992
 No. of 1-blocks: 61, 2-blocks: 12, max blk sz: 126, max free sz: 131
GC memory layout; from 200001f0:
00000: h=BTBBBBBBBhSShhBBBBLhLhhh=h=h=hBhh=h=hhhhhhhhh=Bhh..hhhh==h=hhS
00400: SSh=======h=====h=ShSh==h=h=====SSh=h==Sh=====SSh=S..h==h==h====
00800: ====h=======h==========h=====h====h=======h====h====h=====h=====
00c00: ======.....h=======.....h============h==========================
01000: ================================================================
01400: ===================================.............................
01800: .............h===============h=========h====h======h===hh=======
01c00: ====hh==hh==hh====hh===h==hh==hh=================...............
02000: ................................................................
02400: ....................................................
```

What is garbage collection?

- Memory allocations are checked for references
- Once all references are gone, block is de-allocated
- Circular references cause problems!



```
GC: total: 10048, used: 7056, free: 2992
 No. of 1-blocks: 61, 2-blocks: 12, max blk sz: 126, max free sz: 131
GC memory layout; from 200001f0:
00000: h=BTBBBBBBBhSShhBBBBLhLhhh=h=h=hBhh=h=hhhhhhhhh=Bhh..hhhh==h=hhS
00400: SSh=======h=====h=ShSh==h=h=====SSh=h==Sh=====SSh=S..h==h==h====
00800: ====h=======h==========h=====h====h=======h====h====h=====h=====
00c00: ======.....h=======.....h============h==========================
01000: ================================================================
01400: ===================================.............................
01800: .............h===============h=========h====h======h===hh=======
01c00: ====hh==hh==hh====hh===h==hh==hh=================...............
02000: ................................................................
02400: ....................................................
```

How does software use hardware?

- Device drivers handle the details
- Specific to each hardware model
- Must be provided by manufacturer
- Common devices have generic drivers/distributed via operating systems

Ok, but how do we use device drivers?

- Operating systems require standard interfaces – application programming interfaces (*APIs*)

- This concept is also used for allowing other software to inter-operate

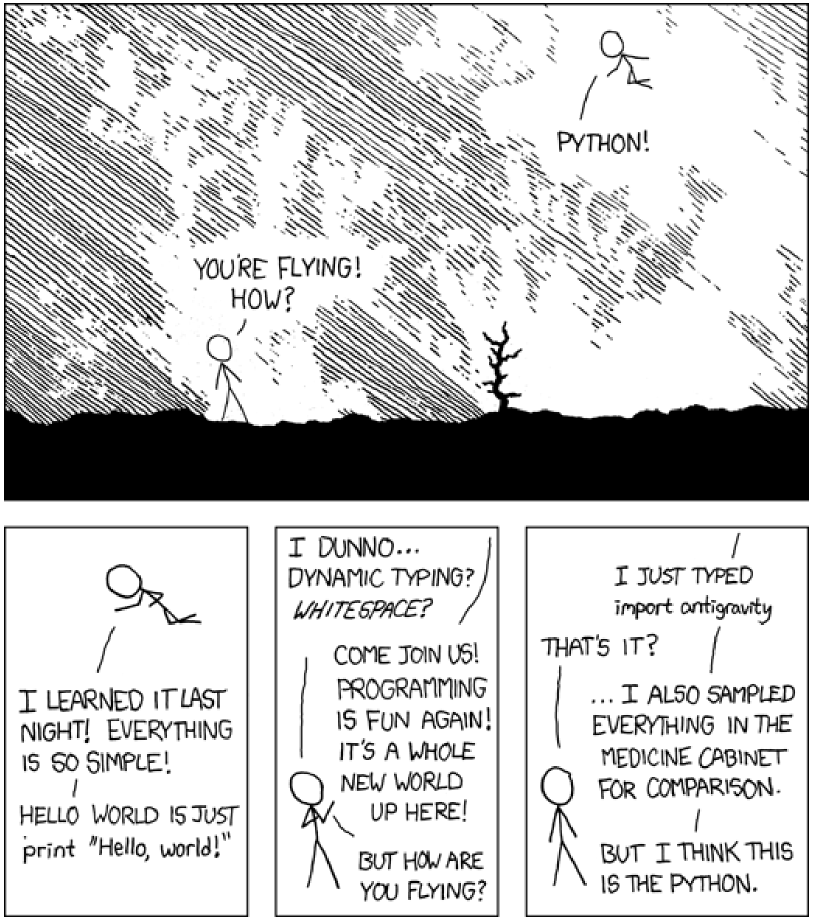
Python imports device drivers.

- Python imports are the way to access APIs
- Help files and contextual help provide API documentation
- APIs also are used to access other software
    - Google maps
    - Financial systems
    - Cloud storage
    - etc.

```
from microbit import button_a
if button_a.was_pressed():


from microbit import display
display.on()
display.clear()
display.show('A')
```

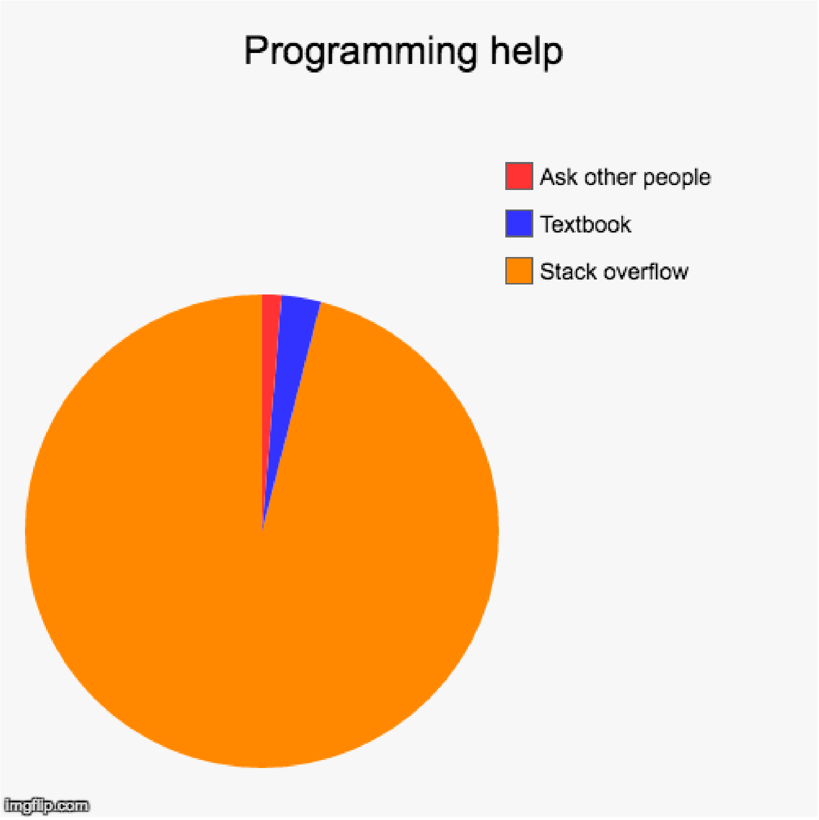
Python can import many things!


YOU'RE FLYING! HOW?
PYTHON!
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

- Helpful libraries:
- `import numpy as np`
- `import matplotlib.pyplot as plt`
- Other software:
- `from git import Repo`
- `from selenium.webdriver import Firefox`
- Device drivers:
- `import serial`
- `import mouse`
- Outside services:
- `from bitbucket.client import Client`
- `from pydrive.drive import GoogleDrive`

We can’t teach you all the APIs.

- Self-directed learning is key to successful programming.
- Every API comes with documentation
  - It’s not always good…
- You may write software for others to use
  - You may need to write documentation!



APIs make programming look easy.

```
from github import Github
import csv

g = Github(‘access_token’)
users = []
repos = []
ids = []

for invite in
g.get_user().get_invitations():
    users += [invite.inviter.login]
    repos += [invite.repository.name]
    ids += [invite.id]
for id_value in ids:

g.get_user().accept_invitation(id_value)

with open(‘invites.csv’, ‘a’,
newline=‘’) as f:
    write_out = csv.writer(f)

write_out.writerows(zip(users,repos))
```

- This is how I accept GitHub invitations from an entire class!
- Those two imports harness tens of thousands of lines of other people’s code

Tomorrow: Operating Systems

Image References

Slide 1: *Old Days*, by Randall Munroe, from https://xkcd.com/1755/ (CC BY-NC 2.5)
Slide 5 (left): from https://commons.wikimedia.org/wiki/File:Cloud_computing_icon.svg (CC BY-SA 3.0)
Slide 6: by Daniel Sancho, from https://www.flickr.com/photos/teclasorg/46802073531 (CC BY 2.0)
Slides 7 and 8: by Steven Depolo, from https://www.flickr.com/photos/stevendepolo/3072821281 (CC BY-NC 2.0)
Slides 9 and 10: by Jonathan Joseph Bondhus, from https://commons.wikimedia.org/wiki/File:Stack_of_Copy_Paper.jpg (CC BY-SA 3.0)
Slide 11: from https://commons.wikimedia.org/wiki/File:ProgramCallStack2_en.png (Public domain)
Slide 12: from https://en.wikipedia.org/wiki/File:Windows_NT_BSOD_at_GVA_baggage_claim,_1999-10-03.jpg (CC BY-SA 3.0)
Slide 13: by Perplexeus, from https://commons.wikimedia.org/wiki/File:Haystack.png (CC BY-SA 4.0)
Slide 14: from https://commons.wikimedia.org/wiki/File:Typical_computer_data_memory_arrangement.png (CC BY-SA 4.0)
Slides 17 (top) and 18 (top): from https://commons.wikimedia.org/wiki/File:SanDisk-Cruzer-USB-4GB-ThumbDrive.jpg (Public domain)
Slides 17 (bottom) and 18(bottom): by Dmitry Nosachev, from https://commons.wikimedia.org/wiki/File:Supermicro_AOC-SGP-I2_Gigabit_Ethernet_NIC,_PCI-Express_x4_card.jpg (CC BY-SA 4.0)
Slide 20: *Python*, by Randall Munroe, from https://xkcd.com/353/ (CC BY-NC 2.5)
Slide 21: from https://www.reddit.com/r/ProgrammerHumor/comments/69uof3/programming_stack_overflow/