Adventures in Coding
Cyber-Physical Systems
Day 1

Erik Fredericks, 2018
Homepage: http://efredericks.net
GitHub: https://github.com/ou-sbselab/SECS-SummerCamp

All links to images available in speaker notes

8am - 3pm
Lunch 11-12/12-1
30 min break morning/afternoon

8-9
9-10
1030-11
12-1
1-2
230-3

LIBJPEG-DEV
SENSE-HAT

How this will work

I talk for a little bit
You do a fun demo for a little bit

Key point: **don’t be shy! Ask questions!**

If you are interested…

All class materials are posted to my lab’s GitHub page

https://github.com/ou-sbselab/SECS-SummerCamp

(Your stuff is in adventures-in-coding-1)

Here you can find all the presentations, code, guides, etc.

If you want to work on any of this at home

Raspberry Pi 3B:	~$35.00
Sense Hat:	~$30.00

Or there are numerous kits on Amazon for ~$80 (includes case, cables, etc.)

Day 1 Topics

What are cyber-physical systems

Starting up and using the Pi

Introduction to Python

What are cyber-physical systems?

Cyber-physical systems

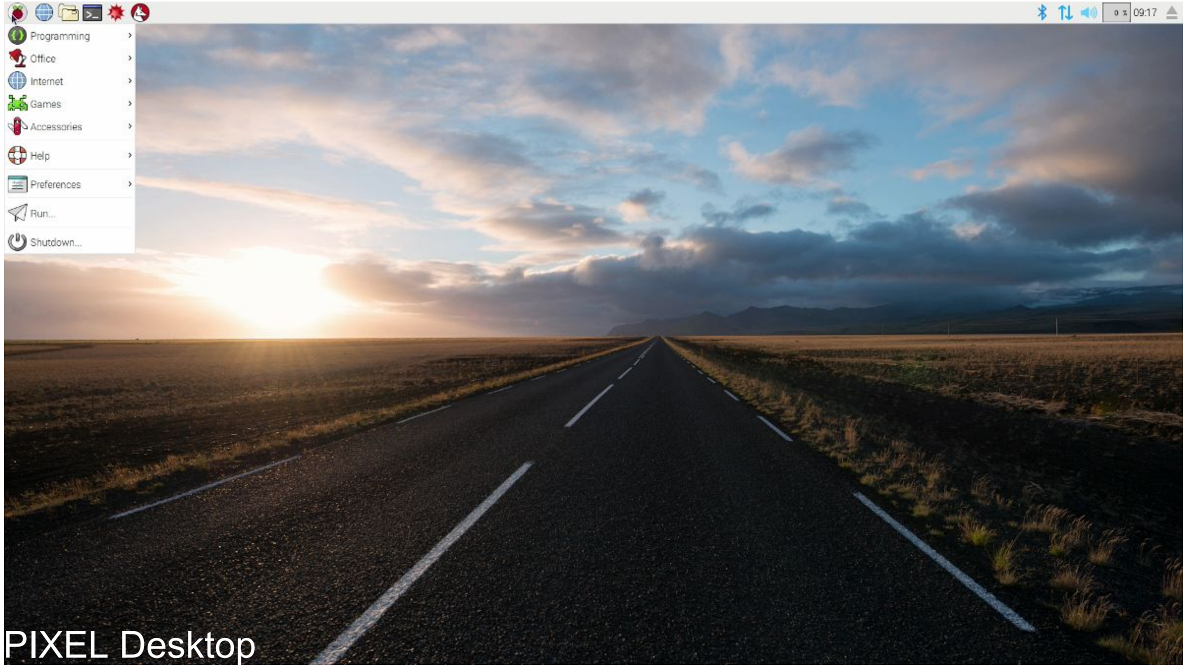
We understand **software**

We understand **hardware**

But, what happens when we put them together?

Starting up and using the Raspberry Pi

1) Plug in the HDMI (video) connection
2) Plug the keyboard and mouse into the USB ports
3) Plug in the power
4) …
5) Success!



PIXEL Desktop

Linux??

Alternative operating system

Can run on nearly any computer system

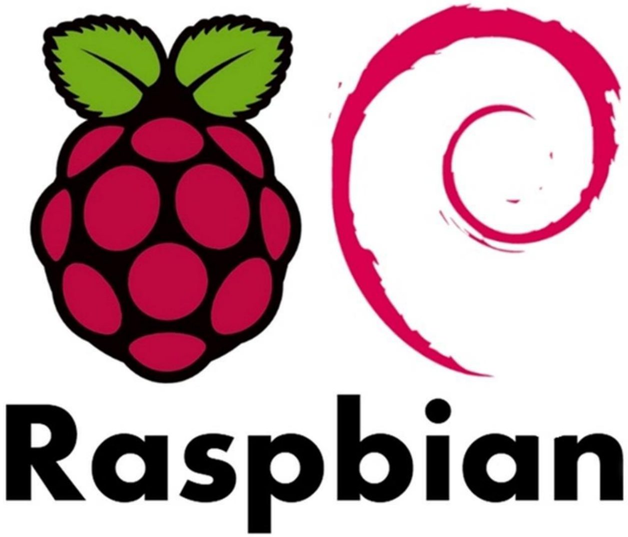
Including Windows now!

Apple?

Linux??

Raspberry Pi?

Raspbian → variant of Debian Linux



Demo 1 - Navigating PIXEL

- Start Menu
- Opening up a document
- Browsing the internet
- Running IDLE

About the demos...

All demos are written in Python

Easy-to-use programming language

(Great starter language if you’re interested in programming)

All files that have a **.py** extension are Python programs

How to run a program called program:

```
$ python program.py
```

Do I need a Raspberry Pi to run Python?

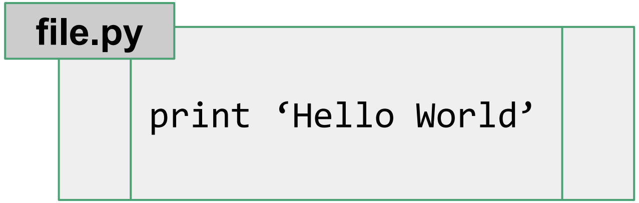
NO

Follow-along code

You can run Python either as a **script** or **interactive**

Script

```
$ python file.py
```



file.py

```
print ‘Hello World’
```

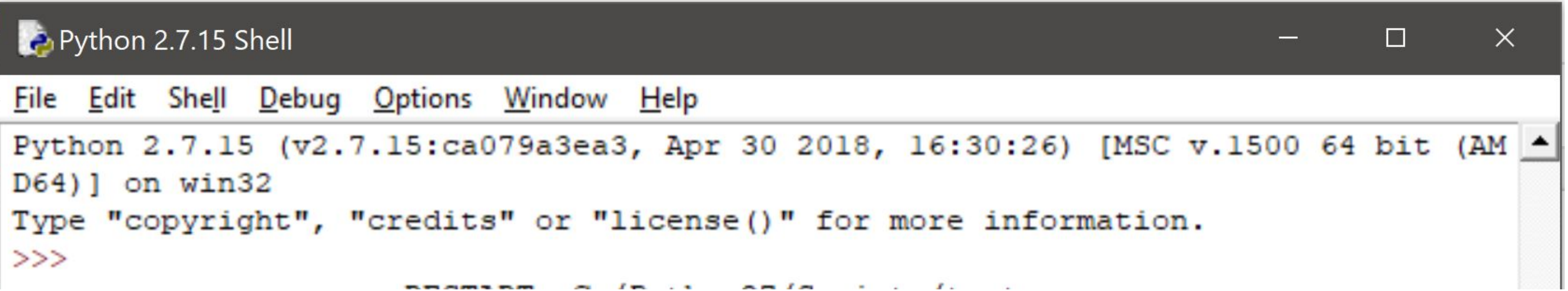
Interactive

```
$ python
>>> print ‘Hello World’
Hello world
>>>
```

Python basics

We’re going to use the **IDLE Python IDE**

Start → Programming → IDLE



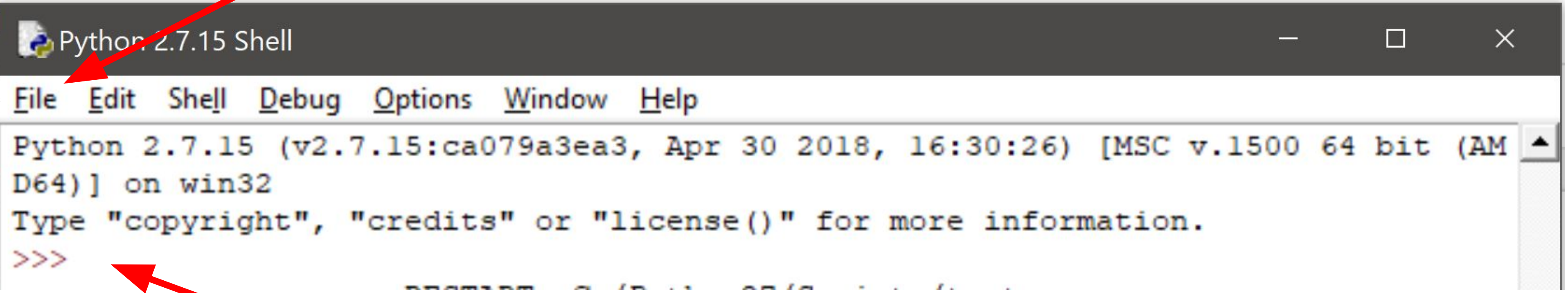
```
Python 2.7.15 (v2.7.15:ca079a3ea3, Apr 30 2018, 16:30:26) [MSC v.1500 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Create script files here

Python basics

We’re going to use the **IDLE Python IDE**

Start ➔ Programming ➔ IDLE



```
Python 2.7.15 (v2.7.15:ca079a3ea3, Apr 30 2018, 16:30:26) [MSC v.1500 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Type Python commands here

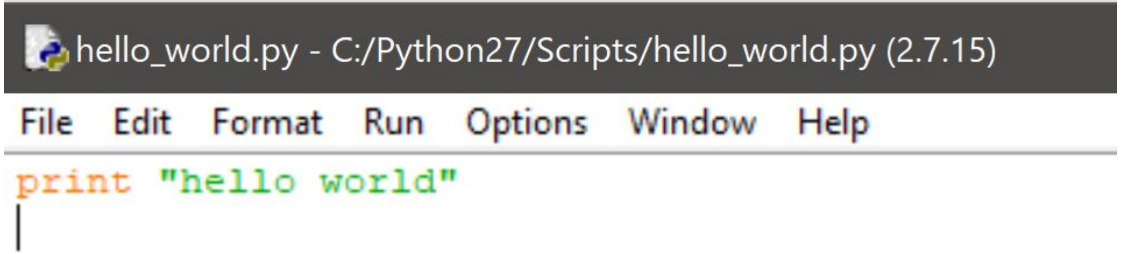
Hello World (your first Python program)

```
>>> print “hello world”
hello world
>>>
```

Hello World (your first Python program)

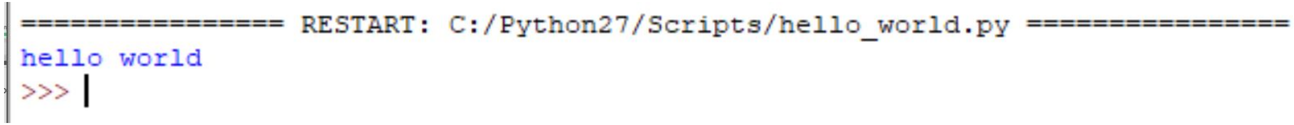
```
>>> print “hello world”
hello world
>>>
```

File → New File
(save as hello_world.py)


hello_world.py - C:/Python27/Scripts/hello_world.py (2.7.15)
File Edit Format Run Options Window Help

```
print "hello world"
```

Run → Run Module (F5)

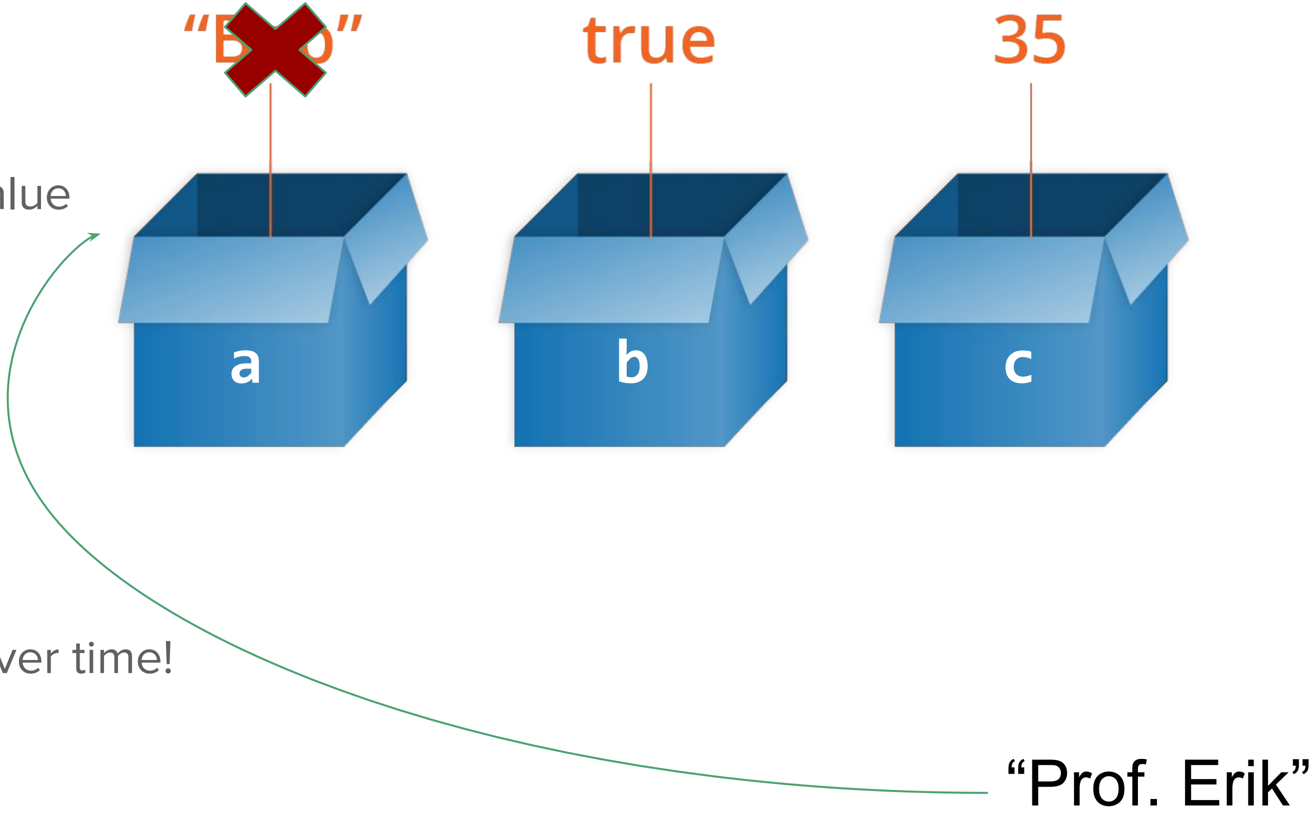


```
================ RESTART: C:/Python27/Scripts/hello_world.py ================
hello world
>>> 
```

hello_world.py

Variables

An object that **holds** a value



This value can change over time!

Variable names

RULES!

Letters, numbers, and underscores (_) all OK!

NO SPACES

Variables can’t start with a number!

What can a variable hold?

Anything really!

- Numbers
- Letters (strings)
- Objects
- ...

Variable basics

Variables have **no type** until you assign them values

```
my_shiny_new_variable = 10

my_shiny_new_variable = “HELLO”

print my_shiny_new_variable
```

Type ➔ is it a number? a string? an object?

Why do we need variables?

Makes our life **so** much easier

Say you're making a game

- How do you keep track of … everything?
  - Player name?
  - What item the player is holding?
  - The color of their boots?

```
playerName  = “Prof. Erik”
Item1       = “HealthUp”
```

What if we want to print variables *and other things*?

This is called concatenation

Combining two **strings**

```
variable_a = “Hello there”
variable_b = “SECS summer camp!”

print variable_a + “ “ + variable_b
```

???

Time to do a thing!

1) In a new script file (File ➔ New File), create a script that:

   a) Has a variable that stores your FIRST NAME
   b) Prints a statement that says:

   ```
   Hello <FIRST NAME>, welcome to OU!
   ```

   (Make sure to replace <FIRST NAME> with your first name)

How to do the thing!

```
my_name = “Prof. Erik”
print “Hello “ + my_name + “, welcome to OU!”
```

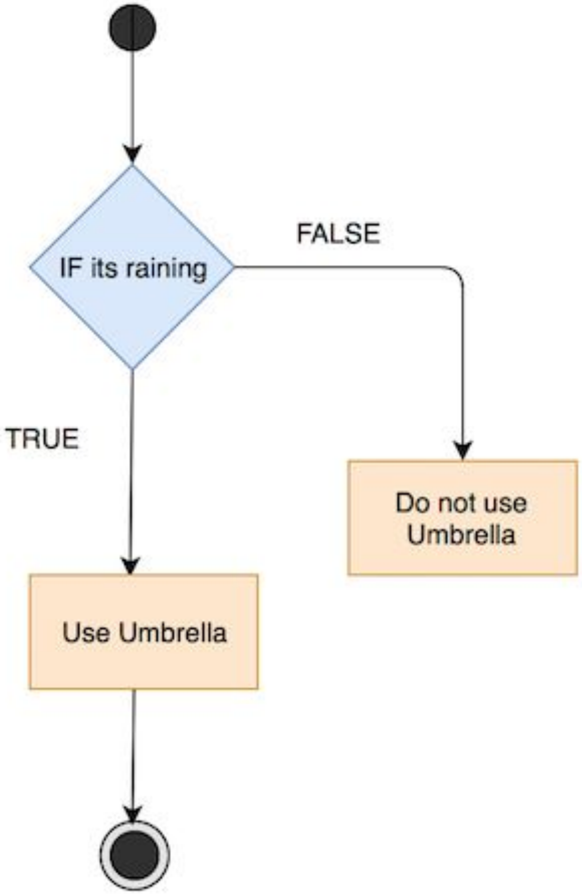
Other ways!

```
output = “Hello %s, welcome to OU!” % (my_name)
```

```
first = “Hello “
second = “, welcome to OU!”
print first + my_name + second
```

Decisions (IF-statements)

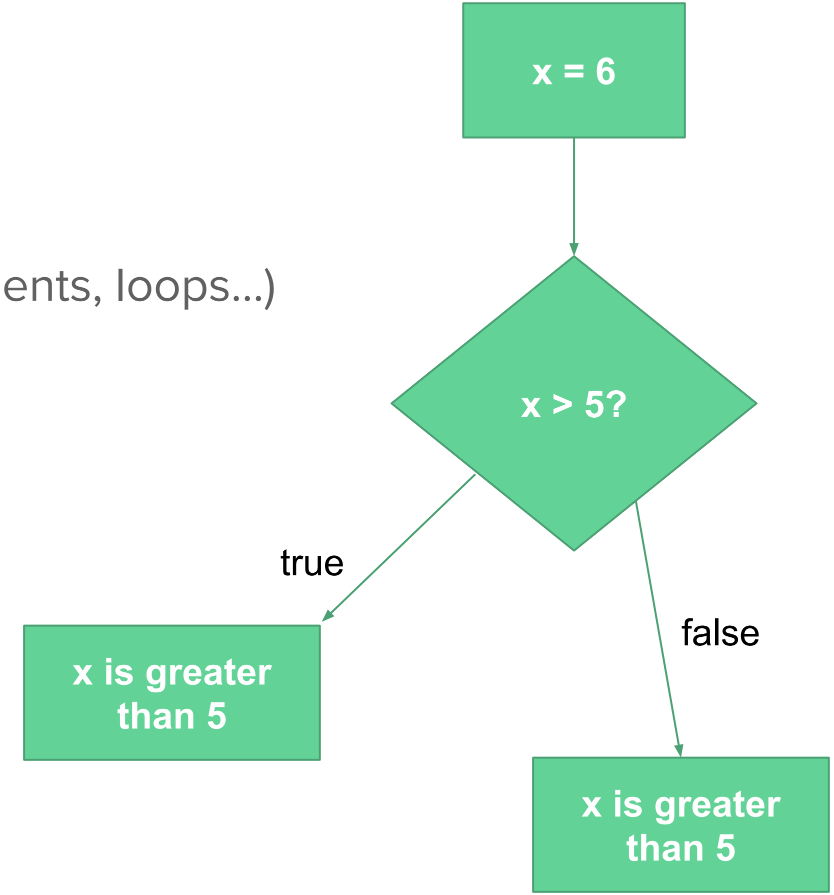
Sometimes you want your program to do things differently



Indenting

Blocks of code work through indents (if statements, loops...)

```
if (x > 5):
  print (‘x is greater than 5’)
else:
  print (‘the else case!’)

-----------------------------
```



Are these the same?

```
if (x > 5):
  print (‘x is greater than 5’)
else:
  print (‘the else case!’)

-----------------------------------------------------------------

if (x > 5):
                          print (‘x is greater than 5’)
else:
                          print (‘is this ok?’)
```

Making things happen over (and over (and over)) - loop

```
x = 0
# Let’s get x to 5
x = x + 1
x = x + 1
x = x + 1
x = x + 1
x = x + 1
```

Loops!

```
x = 0
# for loop
for i in range(0,5):
  x = x + 1
```

range?

`range` is a function that returns a list of numbers

When you say: `range(0, 4)`
  You get back: `[0, 1, 2, 3]`

And when we say: `for i in range(0, 4):`

```
  What is happening is:

i = 0
i = 1
i = 2
i = 3
```

Loop practice!

There are lots of different loop types, but a very common one is the **for** loop

Practice with it!

- Make a **for loop** that prints out the loop variable **twenty times**

```
# For loop reference from earlier
for i in range(0,5):
  x = x + 1
```

for_loop.py

Math!

```
>>> print (3 + 2)

>>> print (5 * 3.5)

>>> a = 5
>>> b = 10
>>> c = a / b
>>> print (c)
```

Game #1: Madlibs

One thing we need to learn first ➔ how to get input from the user!

Enter the **raw_input** function
 ....but what is a function?

Python functions:

```
def my_function():
  print “This is my function.  It is neat.”

my_function
```

Function `raw_input`

Put in a ***:*** *and a space otherwise it will look pretty bad!*

This is built-in to Python (you don’t create it)

```
user_input = raw_input(“Enter some text: ”)
print user_input
```

(Normally we do a check to see if something is there, but for now we can assume the user is handling it OK)

So let’s create a Madlib!

First, our story → ***What I did for summer vacation***

Last summer, my family and I went to a <u>place</u> on vacation. We <u>verb-ed</u> in a <u>vehicle</u>, and it took <u>number</u> days to get there. I took lots of photos of the <u>nouns</u> there, and saw wild <u>animals</u> <u>verb-ing</u> in the <u>noun</u>.

(Snipped to save your typing endurance).

https://www.woojr.com/summer-mad-libs/summer-vacation-madlibs/

Let’s create a Madlib!

Last summer, my family and I went to a place on vacation. We verbed in a vehicle, and it took number days to get there. I took lots of photos of the nouns there, and saw wild animals verb-ing in the noun.

Getting started (you’ll finish the rest)

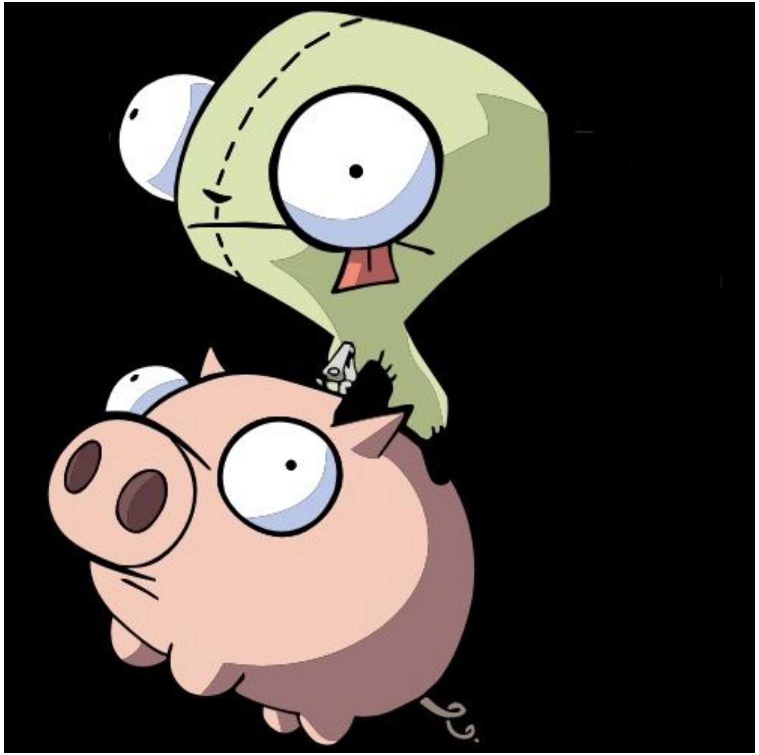
```
place = raw_input(“Enter a place: “)
verb1 = raw_input(“Enter a verb: “)
…
sentence1 = “Last summer, my family and I went to a %s on
vacation.” % (place)
Sentence2 = “We %s in a %s, and it took %s days to get there.” %
(verb1, vehicle, number)
…
```

madlib.py

Game #2: Dice roll

Time to learn how to do random things!

Why is randomness important for programming?



First, import `random`

Our first time **importing** a module!

`import random` ➜ Add extra Python functionality

Random doesn’t exist in our programs until we **import** it

Random number generation

Create a new file: `dice.py`

```
import random
print random.random()
```

Run it a few times and see what happens

Random integers

Last time you generated a `floating` point number (hint: it has lots of decimal points)

Now we want an `integer` (hint: it has no decimal points)

1) Remove `random.random()` from your code
2) Replace it with `random.randint(1,10)`

Run it a few times again

How would you change this to be a die roll?

A quick aside ➜ random.seed

Sometimes you may want to **seed** your random function

This means that your random function will always return the same values each time you run your program

dice_seed.py

Game #3: Guess the dice roll

For this, let’s extend the previous game. Instead of just rolling dice, let’s roll a user-provided number of dice, and then have the player guess what the number is!

How this will work:

1) Ask the player how many dice to roll
2) Roll the dice and store the total value
3) Tell the player the number range to guess on
4) For each guess:
   a) Tell the player if they are higher or lower than the real value
   b) Keep track of the number of guesses

dice_guesser.py

Debugging

public static vo (String[] args) {
try {
File = new Fi "pic.jpg");
long as
= file.lastModifi ();
FileU s.touch( );
long la = file.lastModified
System.out.pri ("File date / tim
astModified1));
1124581

Debugging

Sometimes we have a problem with our code!

One of the easiest and most common ways to fix bugs is **trace debugging**

This means adding **print** statements to see what is going wrong

(If you have a typo, usually the Python interpreter will tell you)

dice_trace.py

Debugging the dice.py file!

Game #4: Adventure

Let’s recovered the fabled amulet of OU in a 3x3 grid. It is randomly placed in the grid and you simply need to explore.

What do we need for this one?

Arrays!

...I mean lists!

- Python calls arrays lists

Game #4 Start:

Make a skeleton of a game:

1) Make a loop that runs forever
2) Inside each loop, ask the user **which direction** to go
3) If the user enters:
    a) **“K” → print “MOVE UP”**
    b) **“J” → print “MOVE DOWN”**
    c) **“H” → print “MOVE LEFT”**
    d) **“L” → print “MOVE RIGHT”**

Arrays

Variables hold **one** of something

```
x = 1
```

Arrays hold **n** of something!

```
y = [1, 2, 3, 4, 5]
print y
```

Arrays

Anybody know what this is?

Working with arrays

```
# never forget your ROY G. BIV
colors = [“red”, “orange”, “yellow”, “green”, “blue”, “indigo”, “violet”]
```

How would you print orange from that list? Red?

```
print colors[1]
print colors[0]
```

Sometimes you might want more than just a list!

What if you wanted a 2-D object? Or 3-D? OR 4-D??!?!

Let’s stick with 2-D.

2-D Array (Lists)

```
two_d_array = [[1,  2,  3,  4,  5],
               [6,  7,  8,  9,  10],
               [11, 12, 13, 14, 15]]
```

How do we reference this?

➡ two_d_array[row][column]

So if I wanted to print out 9....

```
print two_d_array[1]
print two_d_array[1][2]
```

Game skeleton 2 ➜ Make the dungeon

1) Make a 2-D array of your rooms that has 4 columns and 3 rows

Remember the syntax is:

```
My_array = [[1, 2, ...],
            [         ],
            [         ]]
```

2) Print out the item in the 3rd row and 2nd column
   a) Don’t forget arrays start at 0!

What about those loops?

Helpful with lists! Let’s print **all** the colors with **none** of the effort!

```
colors = [“red”, “orange”, “yellow”, “green”, “blue”, “indigo”,
“violet”]

for color in colors:
  print color

## OR

for i in xrange(len(colors)):
  print colors[i]
```

Now, let’s make more of a game!

How about one of those fancy text adventure games from the 80’s?

Things we need:

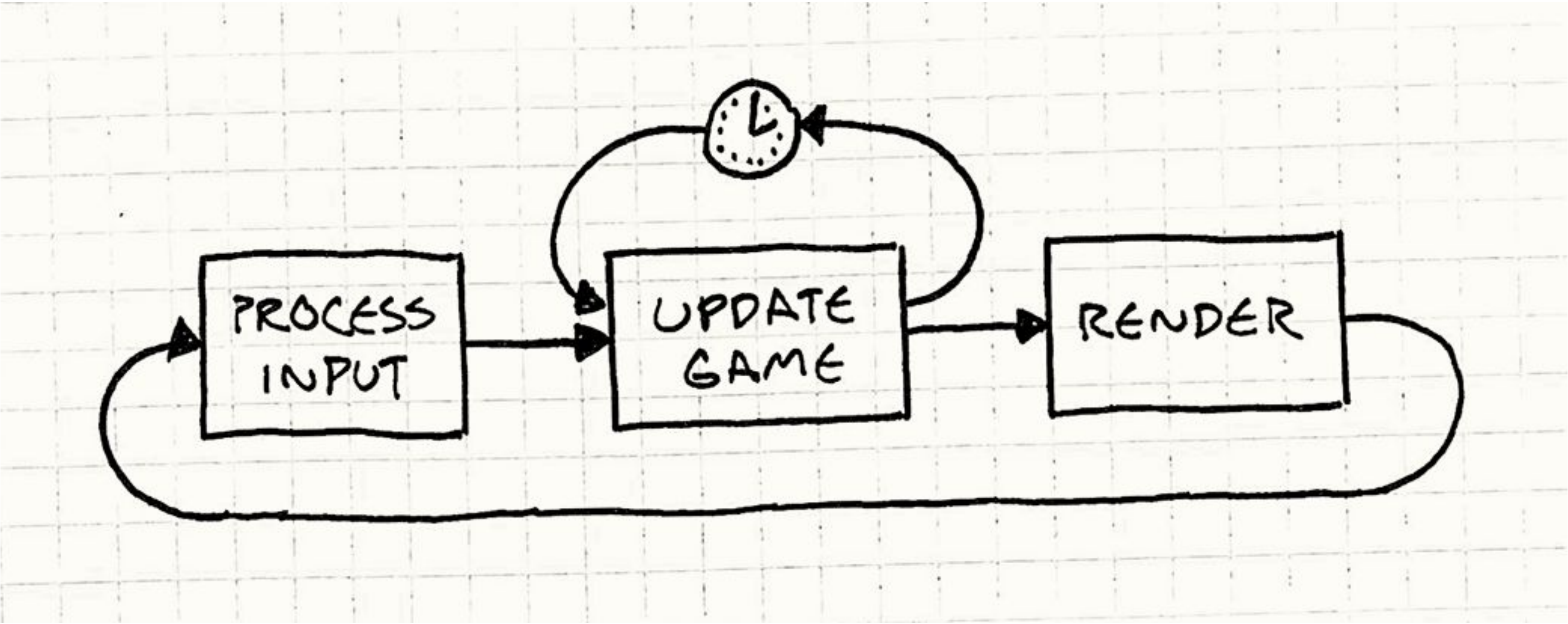
- A 2-dimensional **map** of the game
- Some sort of random **object** to find
  - In this case, we’ll randomly generate a place for it to spawn
- A player to move around

Things we’re not worrying about right now:

- Drawing! Let’s just use text for now
  - If you are interested, look up **pygame**

An aside: game loops

Interested in game programming? LOVE THE LOOP

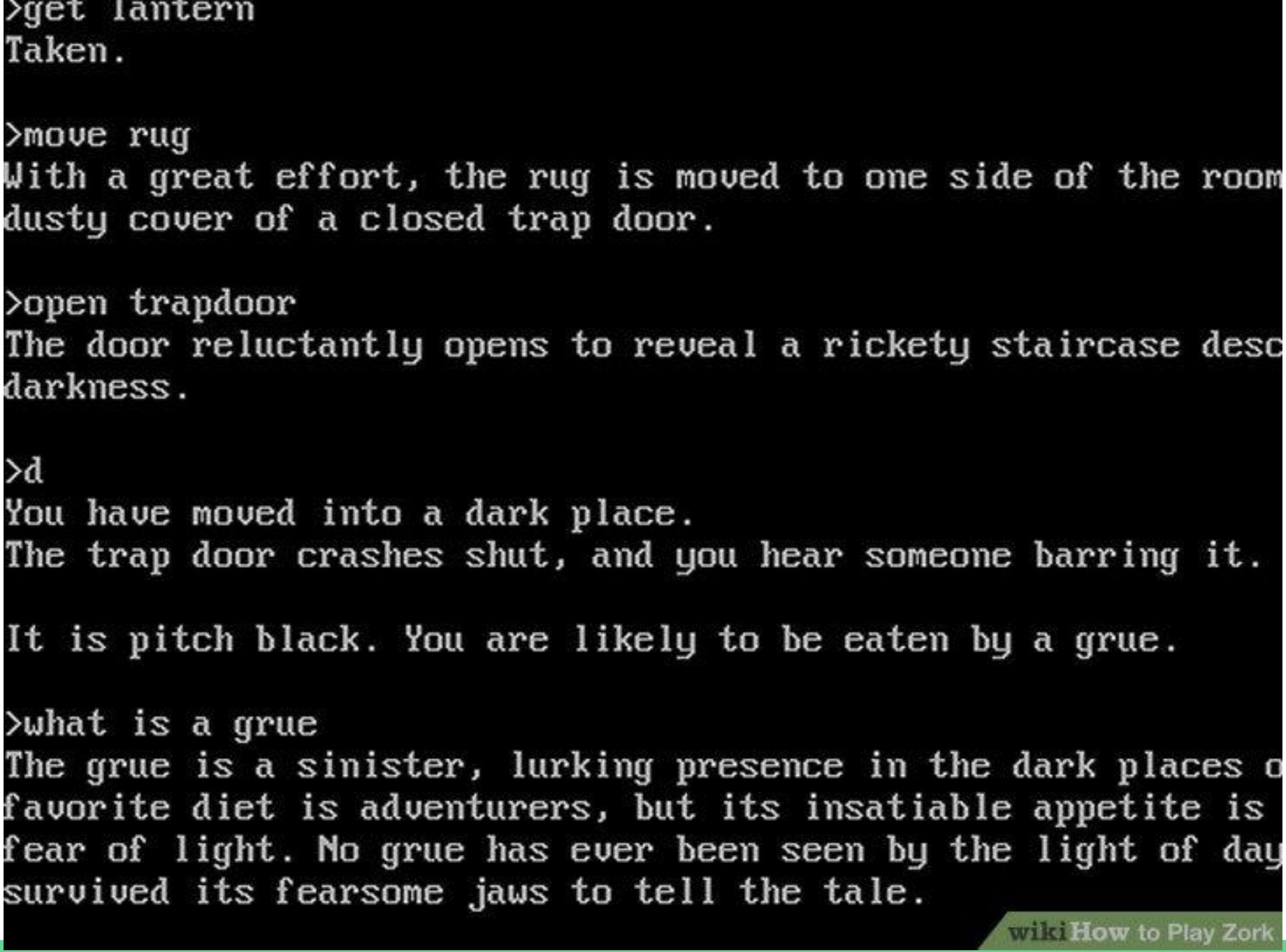


Ok, so our objective

Rescue the AMULET OF OAKLAND UNIVERSITY from the deep, dark, dank dungeon

(Use your imagination … that's what I had to do growing up)

Zork?



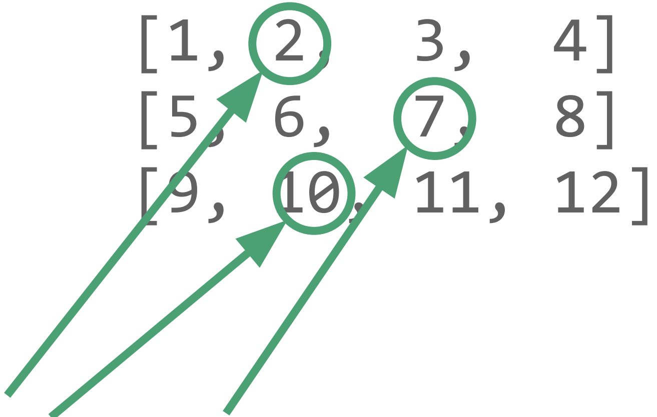
```
>get lantern
Taken.

>move rug
With a great effort, the rug is moved to one side of the room
dusty cover of a closed trap door.

>open trapdoor
The door reluctantly opens to reveal a rickety staircase desc
darkness.

>d
You have moved into a dark place.
The trap door crashes shut, and you hear someone barring it.

It is pitch black. You are likely to be eaten by a grue.

>what is a grue
The grue is a sinister, lurking presence in the dark places o
favorite diet is adventurers, but its insatiable appetite is
fear of light. No grue has ever been seen by the light of day
survived its fearsome jaws to tell the tale.
```

Let’s simplify

Move with four keys:

```
        J
    H       L
        K
```

And we’ll have a 3x4 dungeon



```
[1, 2,  3,  4]
[5, 6,  7,  8]
[9, 10, 11, 12]
```

And we’ll randomly hide the **amulet** at a random spot **each time** you launch the game!

Easy win situation

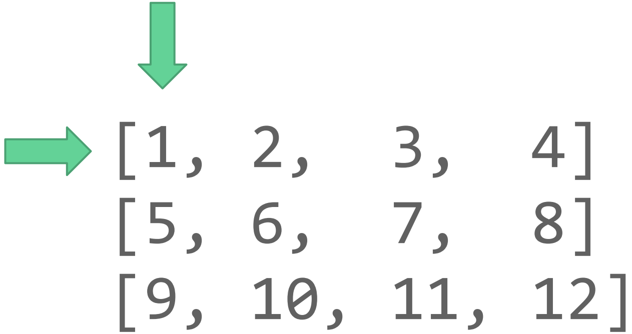
If you enter the room you the amulet is in, you win!
Otherwise, you’re doomed to wander....FOREVER

Math time!

Move DOWN:	J
Move UP:	K
Move LEFT:	H
Move RIGHT:	L

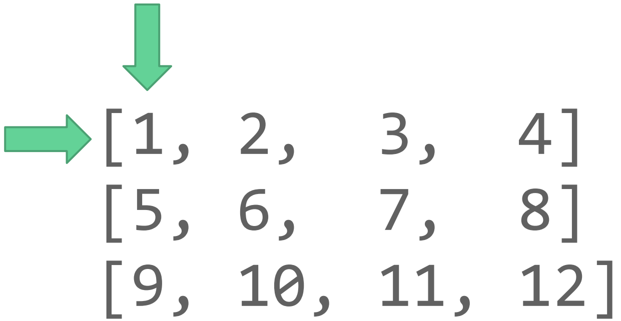
How can we move around? → Math!

Moving **vertically** → update ROW
Moving **horizontally** → update COLUMN



```
[1, 2,  3,  4]
[5, 6,  7,  8]
[9, 10, 11, 12]
```

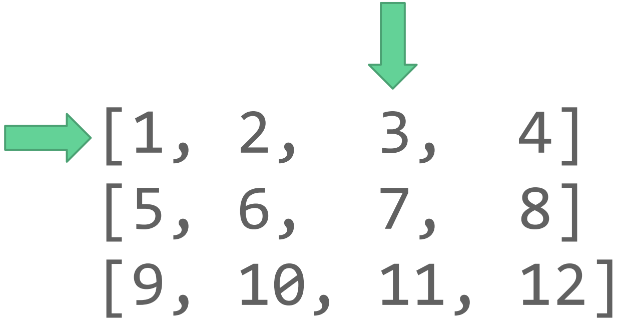
Moving visually



```
[1, 2,  3,  4]
[5, 6,  7,  8]
[9, 10, 11, 12]
```

Row: 0
Column: 0

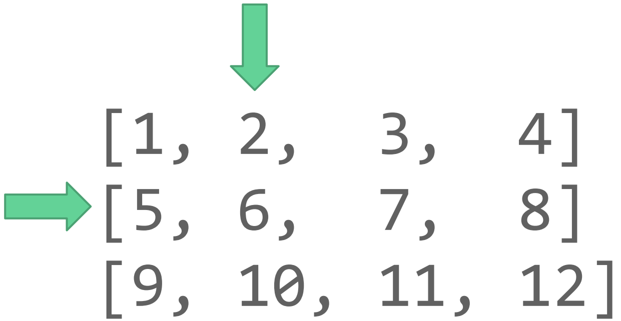
Moving visually



```
[1, 2,  3,  4]
[5, 6,  7,  8]
[9, 10, 11, 12]
```

Row: 0
Column: 2

Moving visually

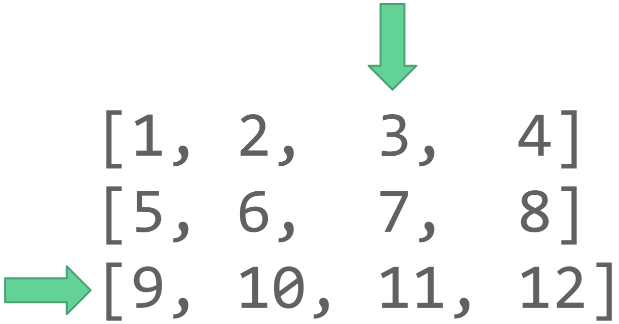


```
[1, 2,  3,  4]
[5, 6,  7,  8]
[9, 10, 11, 12]
```

Row: 1

Column: 1

Moving visually



```
[1, 2,  3,  4]
[5, 6,  7,  8]
[9, 10, 11, 12]
```

Row: 2

Column: 2

Moving

So if we want to move **down**

- Add 1 to our **row**

And **up**?

- Subtract 1 from **row**

And **right**?

- Add 1 to **column**

And **left**?

- Subtract 1 from **column**

So if we want to move **down**

- `current_row += 1`

And **up**?

- `current_row -= 1`

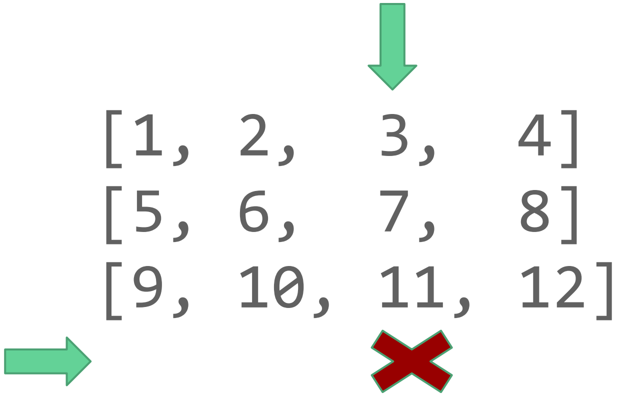
And **right**?

- `current_column += 1`

And **left**?

- `current_column -= 1`

What issue could we have with this?



```
[1, 2,  3,  4]
[5, 6,  7,  8]
[9, 10, 11, 12]
```

Row: 3

Column: 2

Putting it ***all*** together

What could we add?

Enemies?

Maps?

More randomness?!

How about graphics...

pygame

Make a black window (hello world of graphics)

```
import pygame
pygame.init()
screen = pygame.display.set_mode((400, 300))

done = False
while not done:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      done = True

  pygame.draw.rect(screen, (0,128,255), pygame.Rect(30,30,60,60))
  pygame.display.flip()
```

Events

Events are *things* that happen

- Closing the window
- Handling the mouse
- Keyboard

Handle in the event pipeline
→ `for event in pygame.event.get():`

Handling keyboard

```
…
for event in pygame.event.get():
  if event.type == pygame.QUIT:
    done = True

  if ((event.type == pygame.KEYDOWN) and (event.key == pygame.K_SPACE)):
    …

pygame.draw.rect(screen, (0,128,255), pygame.Rect(30,30,60,60))
pygame.display.flip()
```

How would you CHANGE THE COLOR when you hit space?

Task!

1) Make the color change when you hit SPACE

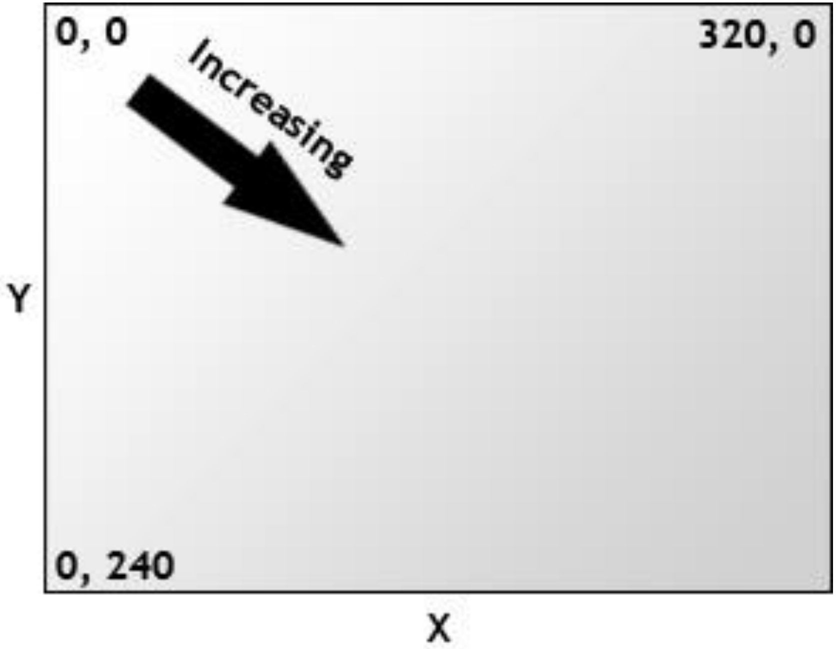
2) Make the color change EVERY LOOP

3) Make the game exit when you hit escape (K_ESCAPE)

Drawing to the screen

Back to the rectangle...

The monitor ➜



Preview of tomorrow

Today we learned **Python** and how to use the **Raspberry Pi** as a computer

Tomorrow, we’re going to learn about monitoring the world around us with the **Pi**

If time allows...

Let’s make some websites!

First off, install the **Apache** web server on your Raspberry Pi.

Open up a **terminal**

Type in:

```
sudo apt-get install apache2
```

day 2 - reality

The Hat

Sense Hat

8x8 RGB LED Matrix

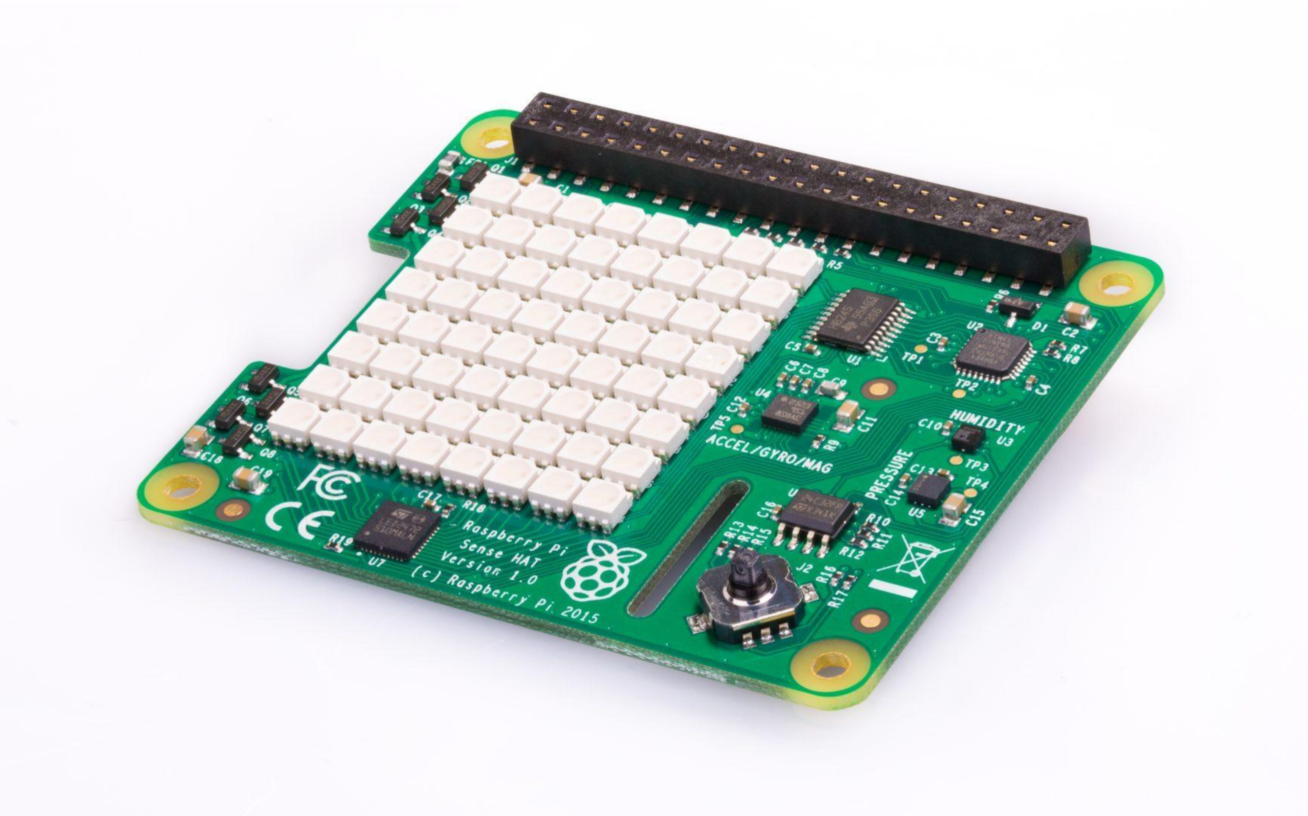
Joystick

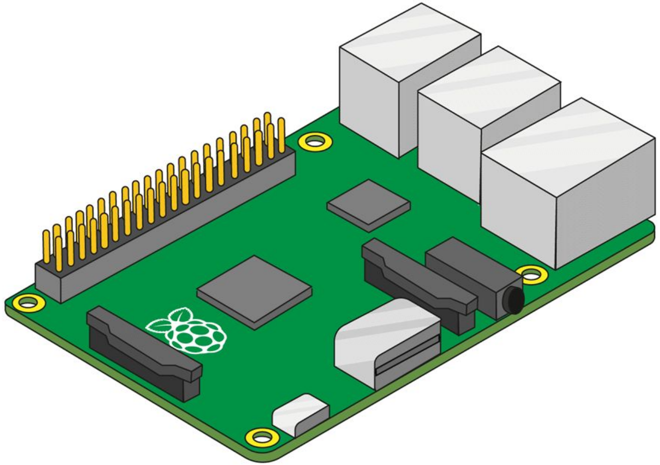
Accelerometer

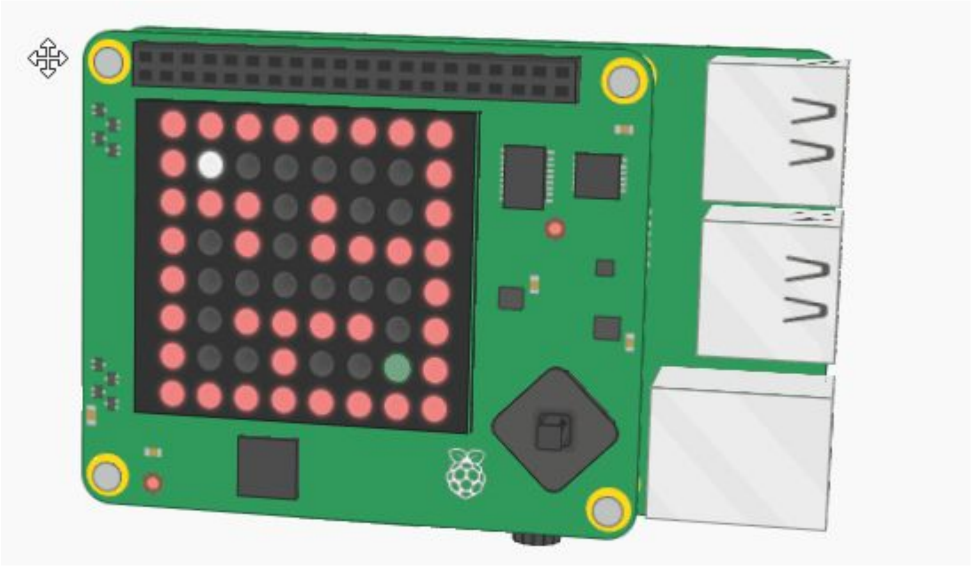
Temperature

Barometric pressure

Magnetometer



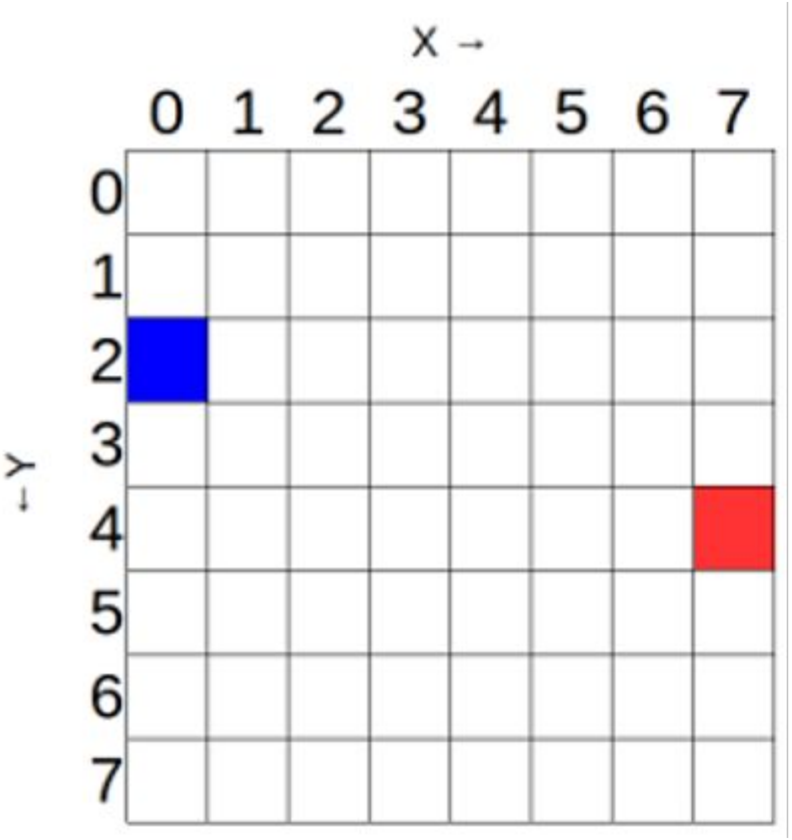




Demo: Temperature

Demo: Temperature on LED matrix

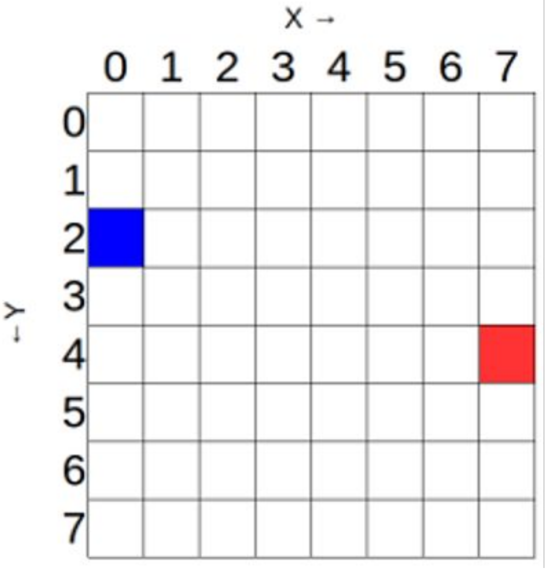
LED Matrix



	0	1	2	3	4	5	6	7
0								
1								
2	🟦							
3								
4								🟥
5								
6								
7								

LED Matrix

```
from sense_hat import SenseHat
sense = SenseHat()

red   = (255,0,0)
blue  = (0,0,255)
green = (0,255,0)

sense.set_pixel(0, 2, blue)
sense.set_pixel(7, 4, red)
```



Sensors