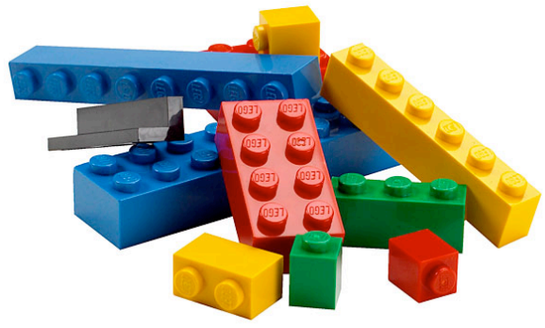
Lecture 2 - Stackoverflow et al.

Spandan Madan
Harvard University

Last Lecture - Recap

- Code is made of little blocks put together something beautiful

- Lego Block = object :
  - has attributes and actions it can do.
  - Ex: File object: .readlines()



- Algorithm: The order you put them together in to achieve some task
  - Plotting COVID growth in India vs Canada

Some new things we saw

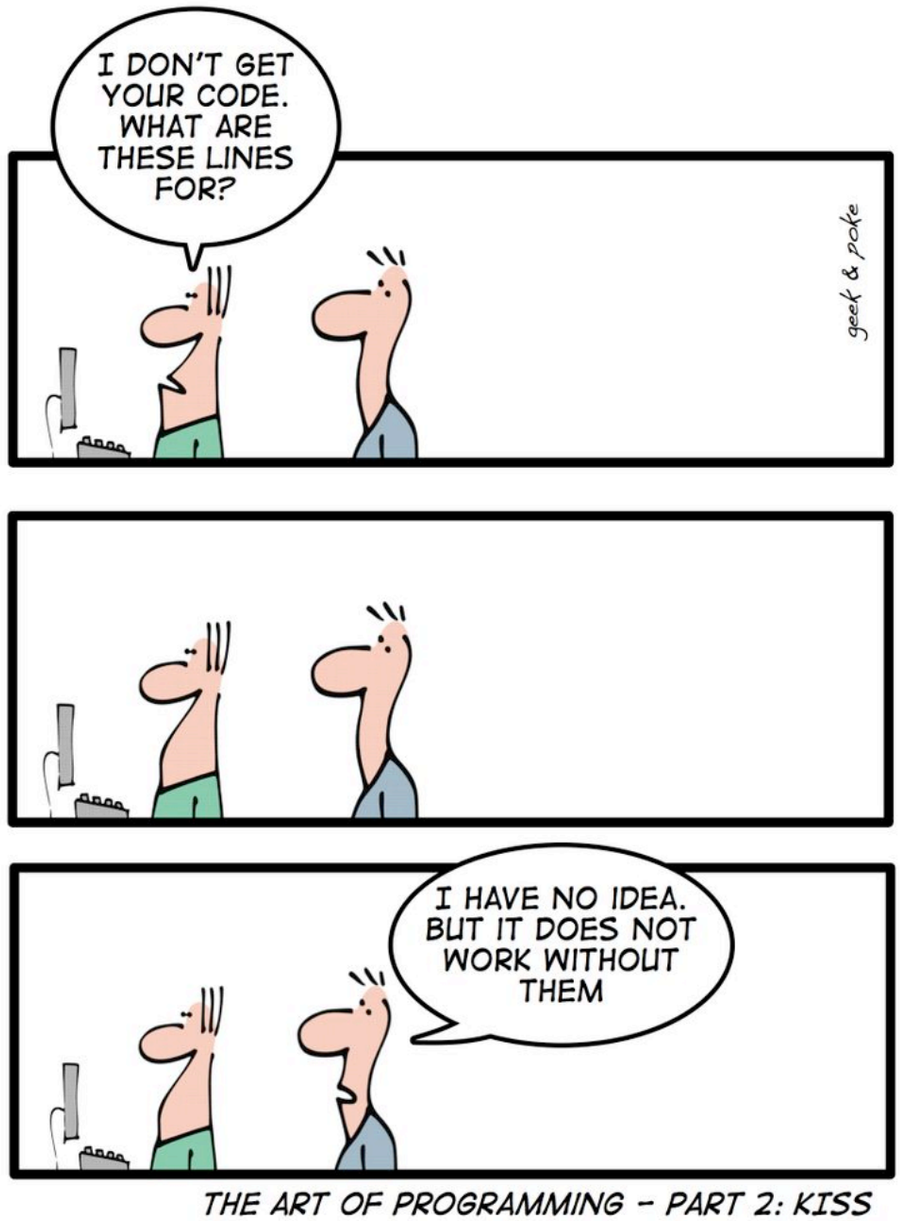
1. **Numbers, Strings (Data Types):**
    - a + b, a - b
    - string_1.split()

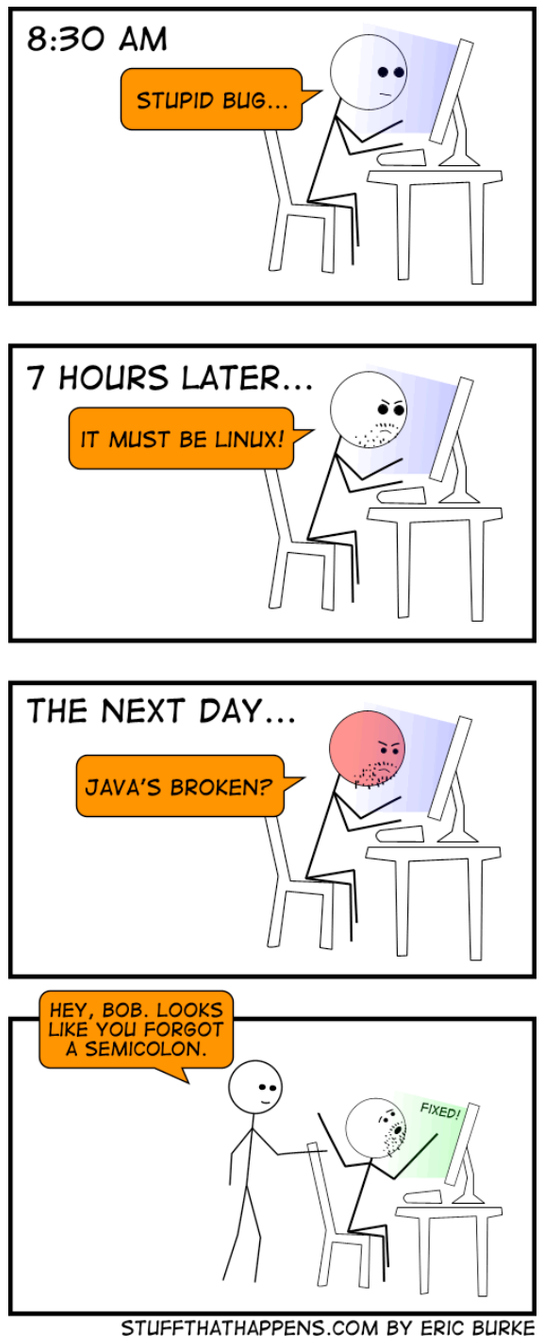
2. **Lists (object):**
    - Examples: [1,3,5], [‘hi’, ‘hello’], [ ], [‘1’,’2’,’3’]
    - append()

3. With open(‘file_path’,’r’) as F:
    - File handler object
    - .readlines()

1. Comments (lines starting with #)
2. Variables (ex: employee_name = ‘Ankit’)
3. Operators:
    - a+b, a-b , a*b, a/b
4. Casting : converting type. Ex: float(‘1’) = 1.0
5. modules: import numpy

The struggles of syntax





How to get better at finding your errors?

- Understand every line that you have written.
- Print everything, literally everything you can.
- Understand how the flow moves from one to another (the imaginary ticker).

Plan for today

1. Understanding how the computer reads your code : the anatomy of code demo.
2. Some new tools for communicating :
    - Dictionary
    - If else
    - Functions

3. *Learning* how to build something

The anatomy of code demo

Dictionaries in Python - I

Pairs of values:

- **Employee name + salary**
- **Country name + population**
- **Firm name + number of employees**

Dictionaries in Python - II

```
list_1 = []
```

```
dictionary_1 = {}
```

Values can also be lists themselves

```
covid_dict = {}

covid['India'] = [10,20,30,40,50]
covid['Canada'] = [12, 24,35,45,60]
```

Accessing information in dictionaries

```
print(list_1[0])
>> 1


for i in range(len(list_1)):
    value_at_i = list_1[i]
    print(value_at_i)
```

```
print(covid['India'])
>> [10,20,30,40,50]


for key in covid.keys():
    value_at_key = covid[key]
    print(key, value_at_key)
```

If else

```
if <condition_is_true>:
    do_this
else:
    do_that
```

```
if <condition_1_is_true>:
    do_thing_1
elif <condition_2_is_true>:
    do_thing_2
else:
    do_that
```

Some examples of conditions

Set variable
a = 1000

```
a = 1000
print(a > 2000)
>> False

print(a < 2000)
>> True
```

Condition

```
a = 1000
if a < 2000:
    print('less than 2000')
```

Lists + for loop + if else + dictionary

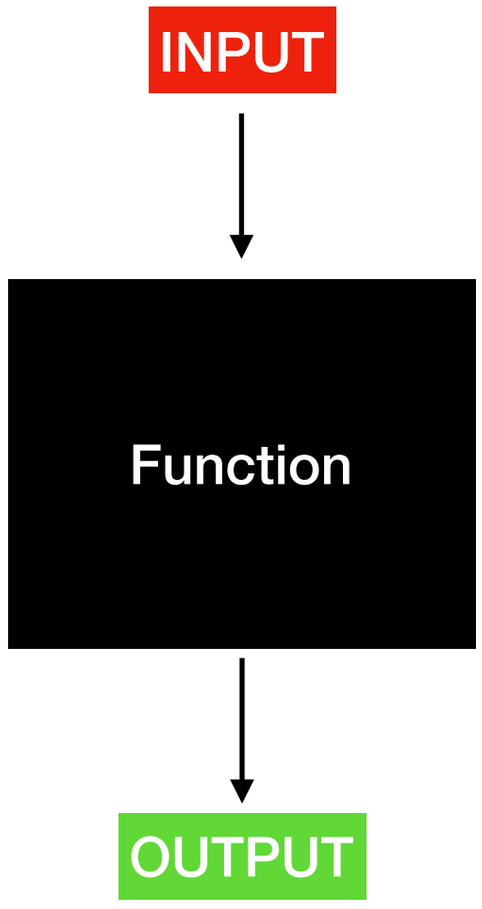
```
# Finding employees with salary > $4,000 per month
# Dictionary employee_salaries already given to us

more_than_4000 = []
less_than_4000 = []

for name in employee_salaries.keys():
    salary = employee_salaries[name]
    if salary > 4000:
        more_than_4000.append(name)
    else:
        less_than_4000.append(name)
```

Functions

- Chunks of re-usable code
- Take inputs and return outputs (most often)



def lets the computer know you're making a function

```python
import numpy as np

# Ques 1 - What is numpy here?
# Ques 2 - What are these 2 lines in red?

# Let's create our first function.
def squared(x):
    squared = x*x
    return squared

print(squared(8))
>> 64
```

squared is the name of the function

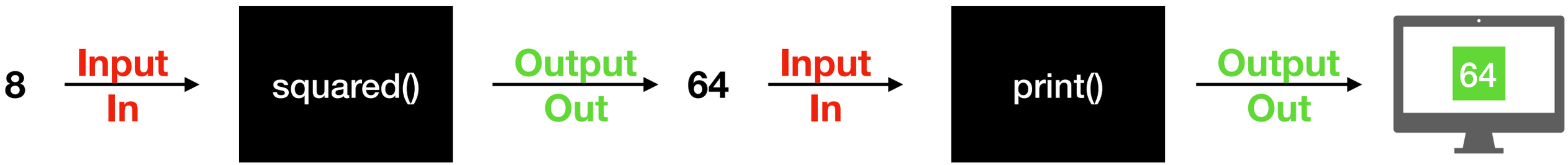
x is the Input. Brackets define the input

return defines the output

```python
def pythagorus(side_1,side_2):
    hyp_squared = squared(side_1) + squared(side_2)
    hypotenuse = np.sqrt(hyp_squared)
    return hypotenuse

print(pythagorus(6,8))
>> 10
```

print(squared(64))



FUNCTION CALL STACK