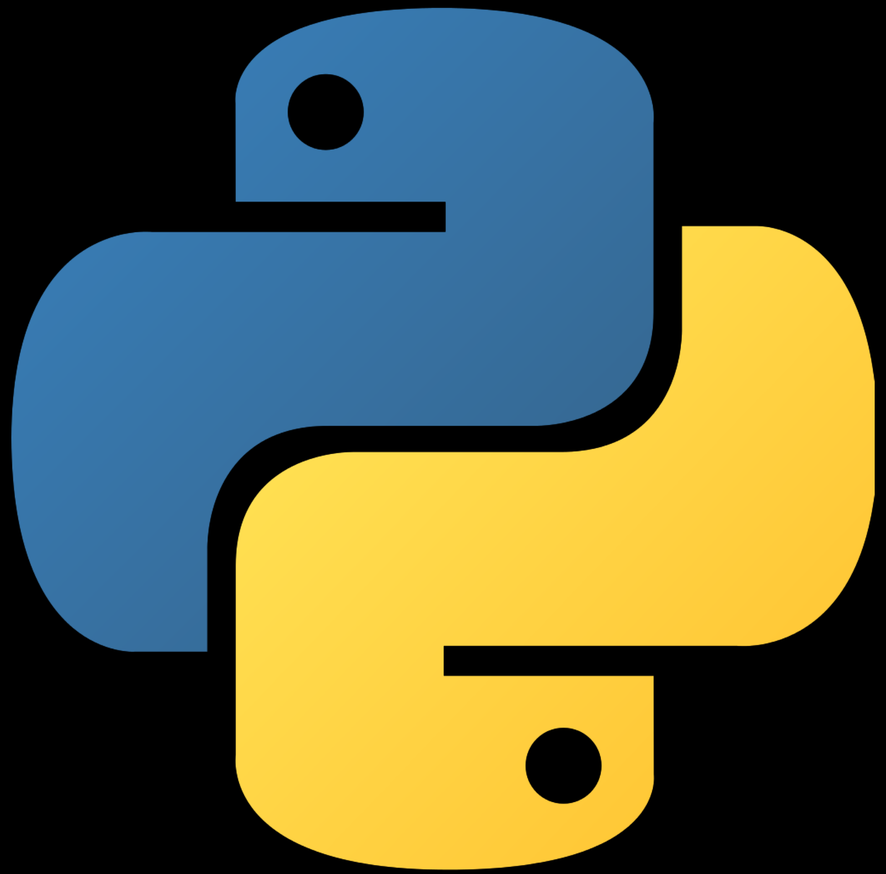
Welcome to Python!

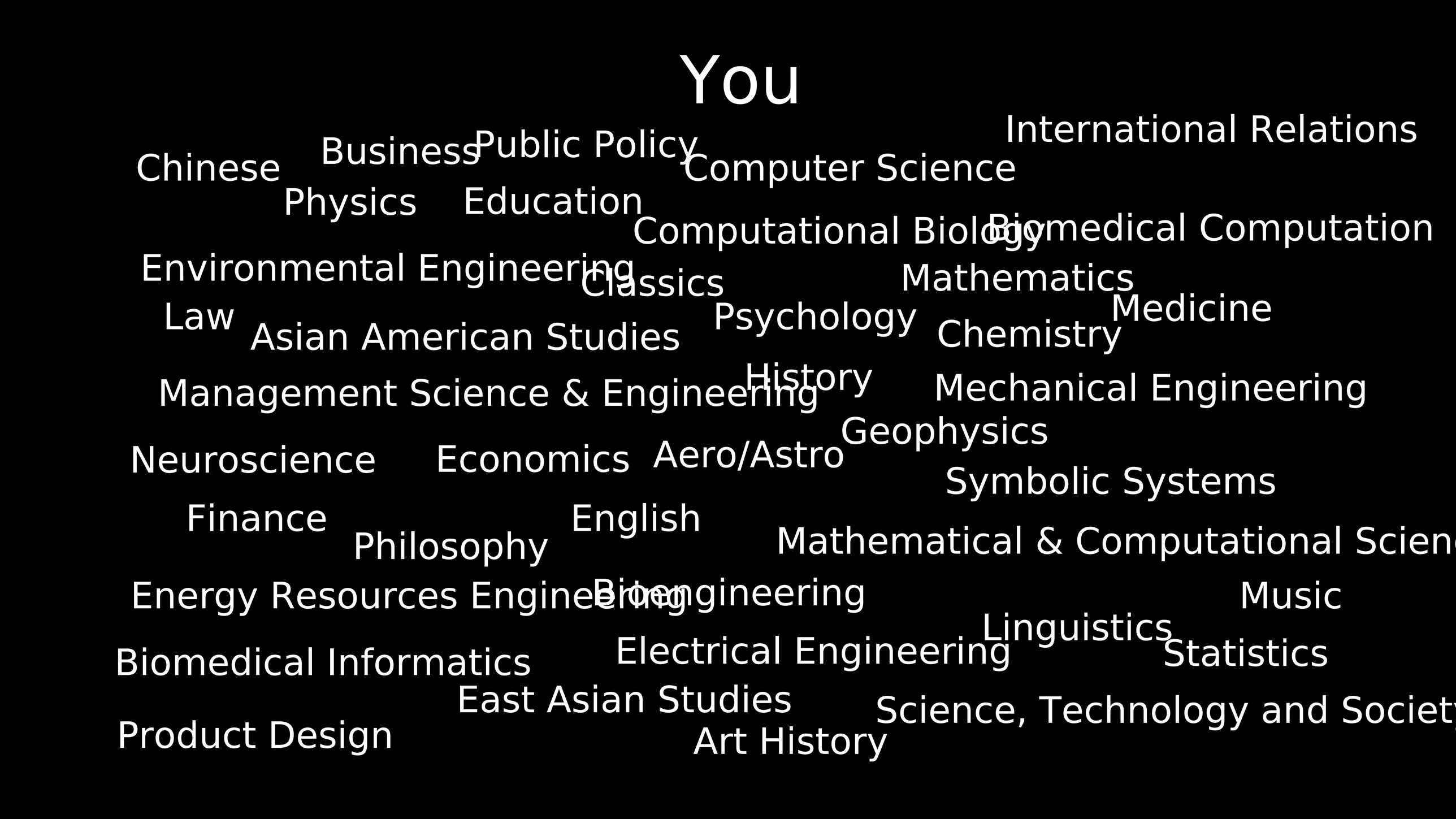
The Python Programming Language



Who am I?

Muhammad Azhdari

Mohammad.azhdari.22@gmail.com


You
International Relations
Business
Public Policy
Chinese
Computer Science
Physics
Education
Computational Biology
Biomedical Computation
Environmental Engineering
Classics
Mathematics
Law
Psychology
Medicine
Asian American Studies
Chemistry
History
Management Science & Engineering
Mechanical Engineering
Geophysics
Neuroscience
Economics
Aero/Astro
Symbolic Systems
Finance
English
Philosophy
Mathematical & Computational Scien
Energy Resources Engineering
Bioengineering
Music
Linguistics
Biomedical Informatics
Electrical Engineering
Statistics
East Asian Studies
Science, Technology and Societ
Product Design
Art History

Course Goals

1. Develop skills with Python fundamentals, both old ar
2. Learn to recognize and write "good" Python
3. Gain experience with practical Python tasks
4. Understand Python's strengths (and weaknesses)

Interpreter VS compiler

Compiler

- Converting the instruction of a high level into machine language.

- Creating object code file.

- program execution is fast.

Interpreter

- interpreting program statement by statement.

- doesn’t create object code file.

- program execution is slow.

Questions

What is Python?

Why Python?

Will Python help me get a job?

History of Python

History of Python

1994

2000

Python 3: 2008

Specifcally, we’re using Python 3.7.2

Philosophy of Python

```
>>> import this
```

The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

>>> import this

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at frst unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

Programmers are more important than programs

“Hello World” in Java

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Yuck

“Hello World” in C++

```
#include <iostream>
using namespace std;


int main() {
    cout << "Hello World!" << endl;
}
```

Double Yuck

“Hello World” in Python

```
print("Hello world!")
```

Who Uses Python?

Python at Stanford

CEE 345: Network Analysis for Urban Systems

COMM 177P: Programming in Journalism

COMM 382: Big Data and Causal Inference

CS 375: Large-Scale Neural Network Modeling for Neuroscience

GENE 211: Genomics

LINGUIST 276: Quantitative Methods in Linguistics

MI 245: Computational Modeling of Microbial Communities

MS&E 448: Big Financial Data and Algorithmic Trading

PHYSICS 368: Computational Cosmology and Astrophysics

PSYCH 162: Brain Networks

STATS 155: Statistical Methods in Computational Genetics

Python in Business

Dropbox

yelp

Eventbrite

venmo

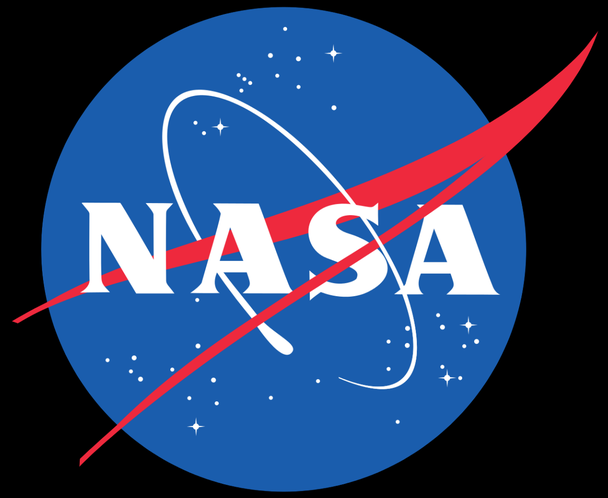
Quora

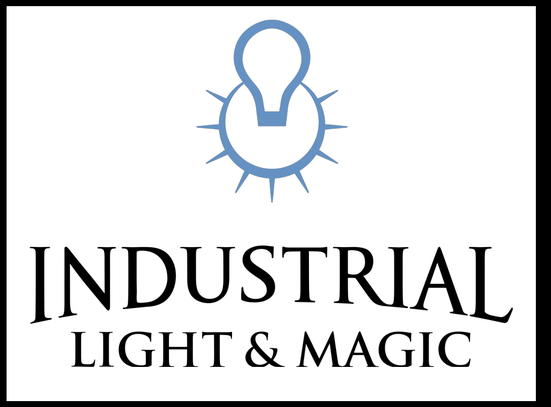
Google

Instagram

YouTube

Other Python Users





3-Minute Break

The Big Picture

The Road Ahead - The Python Langua

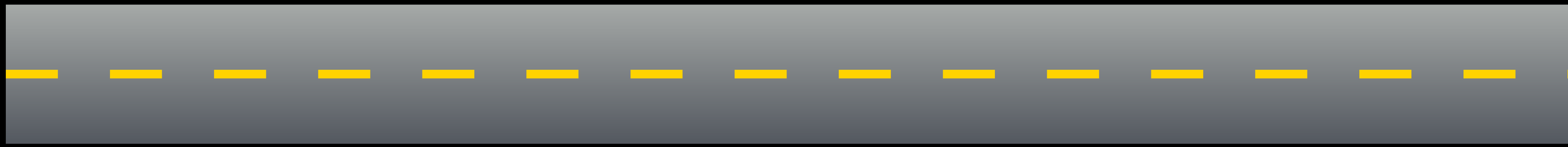
Week 1 Python Fundamentals

Week 2 Data Structures

Week 3 Functions

Week 4 Functional Programming

Week 5 Object-Oriented Python



The Road Ahead - Python Tools

Week 6 Standard Library

Week 7 Third-Party Tools

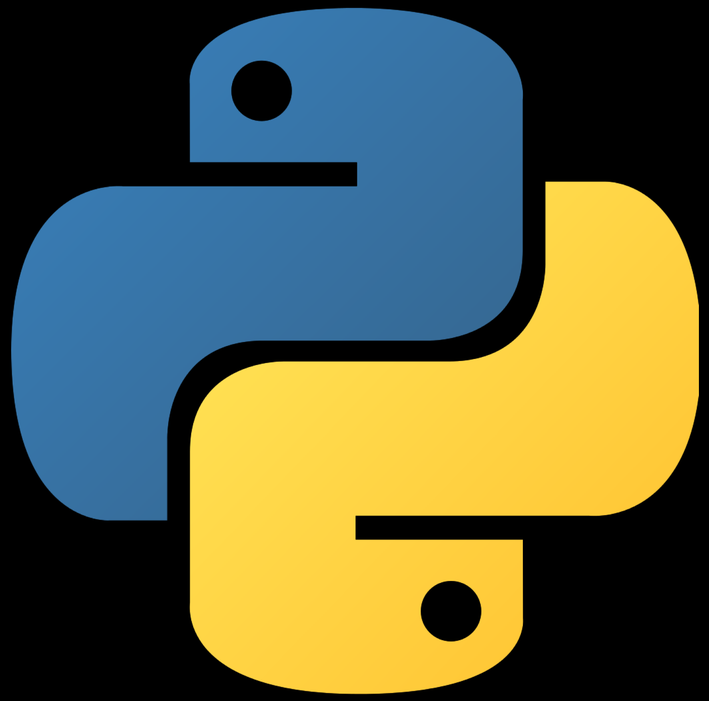
Week 8 Ecosystem

Week 9 Advanced Topics

Week 10 Projects!

Let’s Get Started!

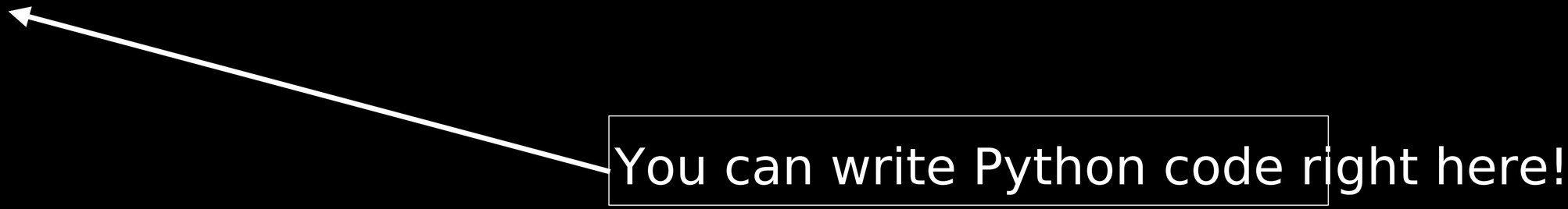
Python Basics



- Interactive Interpreter
- Comments
- Variables and Types
- Numbers and Booleans
- Strings and Lists
- Console I/O
- Control Flow
- Loops
- Functions

Interactive Interpreter

```
sredmond$ python3
Python 3.7.2 (default, Dec 27 2018, 07:35:06)
[Clang 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



A Big Deal

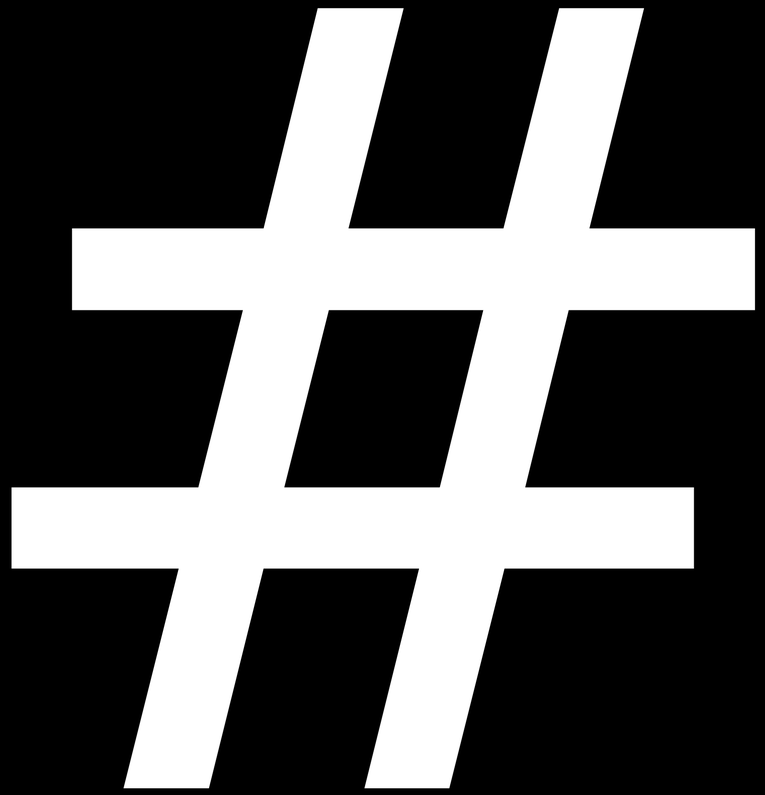
Immediate gratifcation!

Sandboxed environment to experiment with Pyth

Shortens code-test-debug cycle to seconds

The interactive interpreter is your new best friend

Hashtag



Pound Sign

Sharp

Number Sign

Octothorpe

Comments

```
# Single line comments start with a '#'


"""
Multiline comments can be written between
three "s and are often used as function
and module comments.
"""
```

Variables

Variables

```
x = 2
x * 7
# => 14
```

No semicolon!

What happened here?!

```
x = "Hello, I'm "
x + "Python!"
# => "Hello, I'm Python"
```

Where’s My Type?


```
int x = 0   ;
```

Where’s My Type?

Variables in Python are **dynamically-typed**: declared without an explicit

However, **objects** have a type, so Python knows the type of a variable, ev

```
type(1)       # => <class 'int'>
type("Hello")  # => <class 'str'>
type(None)    # => <class 'NoneType'>
```

This is the same object
as the literal type int

```
type(int)     # => <class 'type'>
type(type(int))# => <class 'type'>
```

Python's dynamic type syste
is fascinating! More on Wed

Numbers and Math

Numbers and Math

```
3          # => 3   (int)
3.0         # => 3.0 (foat)

1 + 1       # => 2
8 - 1       # => 7
10 * 2      # => 20
5 / 2       # => 2.5
13 / 4      # => 3.25
9 / 3       # => 3.0
7 / 1.4     # => 5.0

7 // 3      # => 2  (integer division)
7 % 3       # => 1  (integer modulus)
2 ** 4      # => 16 (exponentiation)
```

Python has two numeric type

int and foat

Booleans

Booleans

```
True            # => True
False           # => False

not True        # => False
True and False  # => False
True or False   # => True (short-circuits)

1 == 1          # => True
2 * 3 == 5      # => False
1 != 1          # => False
2 * 3 != 5      # => True

1 < 10          # => True
2 >= 0          # => True
1 < 2 < 3       # => True  (1 < 2 and 2 < 3)
1 < 2 >= 3      # => False (1 < 2 and 2 >= 3)
```

bool is a subtype of int, where

True == 1 and False == 0

Strings

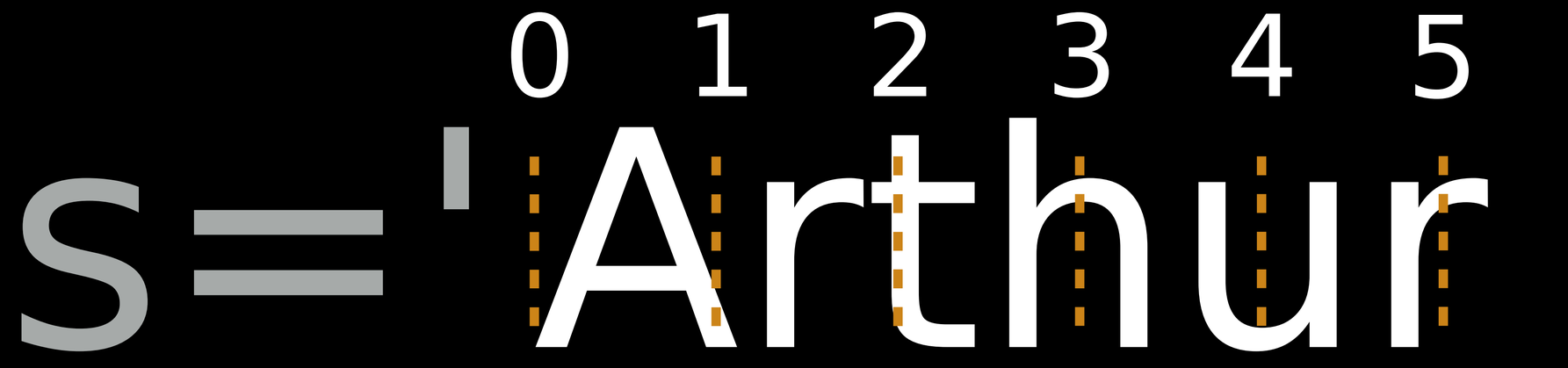
Strings

No char in Python!

Both ' and " create string literals

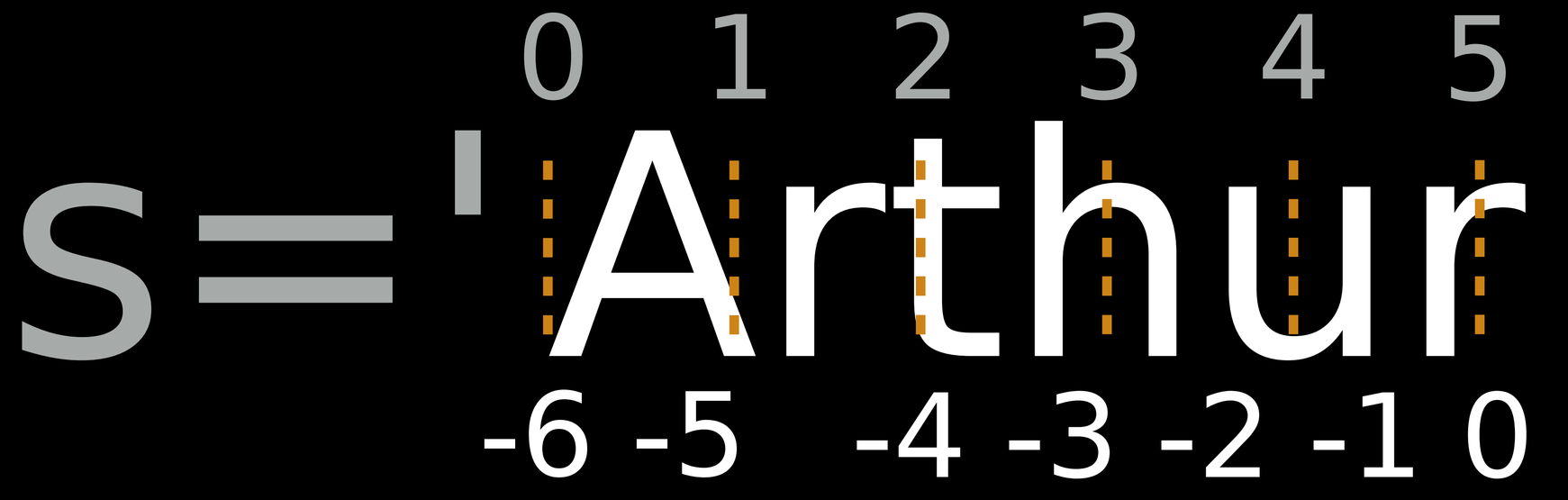
```
greeting = 'Hello'
group = "wørld"  # Unicode by default


greeting + ' ' + group + "!" # => 'Hello wørld!'
```

Indexing




```
s[0] == 'A'
s[1] == 'r'
s[4] == 'u'
s[6] # Bad!
```

Negative Indexing




```
s[-1] == 'r'
s[-2] == 'u'
s[-4] == 't'
s[-6] == 'A'
```

Slicing


```
s[0:2] ==  'Ar'
s[3:6] == 'hur'
s[1:4] == 'rth'
```

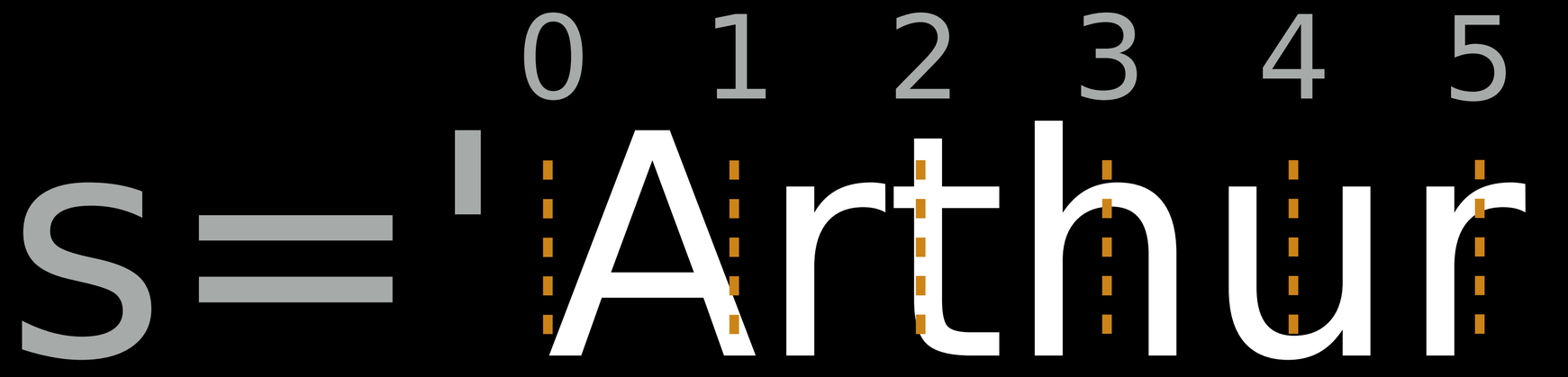
Strings

Implicitly starts at 0

```
s[:2] ==  'Ar'
s[3:] == 'hur'
```

Implicitly ends at the end

Strings



Can also pass a step size

```
s[1:5:2] == 'rh'
s[4::-2] == 'utA'
 s[::-1] == 'ruhtrA'
```

One way to reverse a string

Converting Values

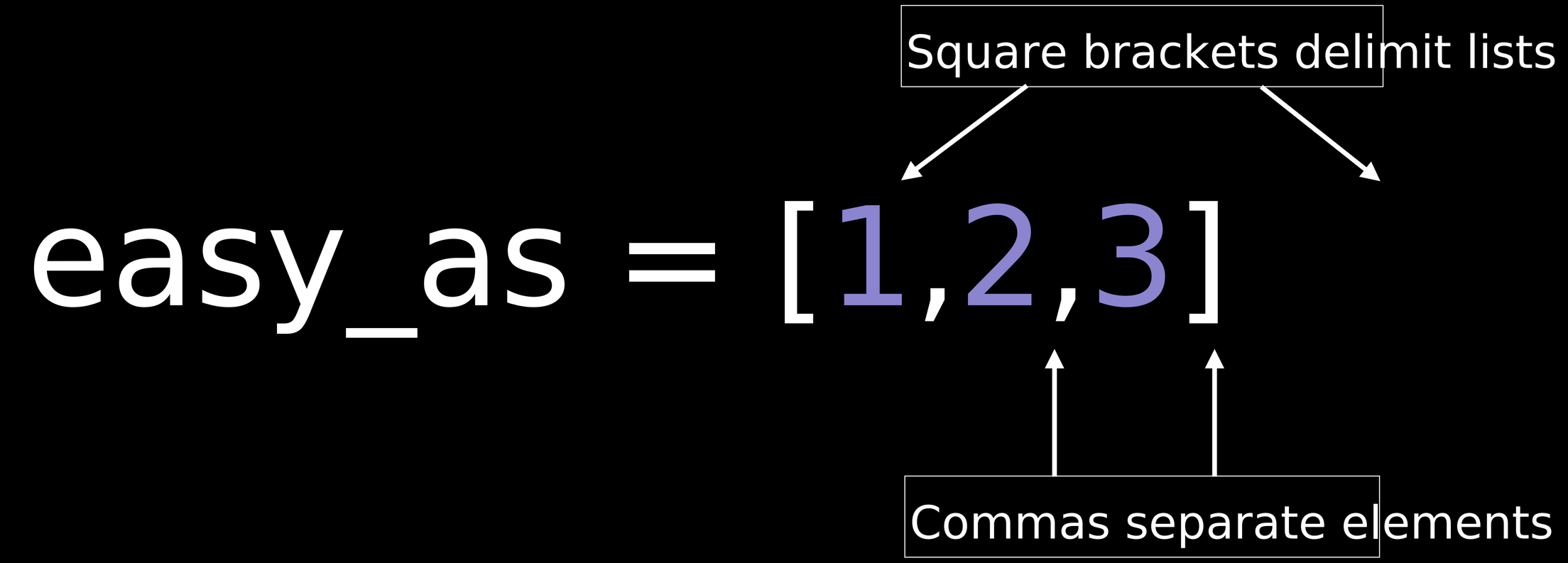
```
str(42)       # => "42"
int("42")     # => 42
foat("2.5")   # => 2.5
foat("1")     # => 1.0
```

All objects have a string representation

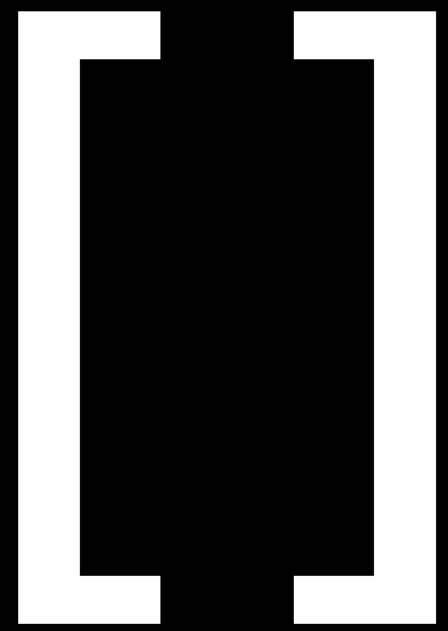
Lists

Dive into Python data structures Week 2!

Lists


Square brackets delimit lists
easy_as = [1,2,3]
Commas separate elements

Lists



Versatile

Incredibly common

≈ ArrayList / Vector

Basic Lists

```
# Create a new list
empty = []
letters = ['a', 'b', 'c', 'd']
numbers = [2, 3, 5]

# Lists can contain elements of diferent types
mixed = [4, 5, "seconds"]

# Append elements to the end of a list
numbers.append(7)   # numbers == [2, 3, 5, 7]
numbers.append(11)  # numbers == [2, 3, 5, 7, 11]
```

Inspecting List Elements

```
letters = ['a', 'b', 'c', 'd']
numbers = [2, 3, 5, 7, 11]

# Access elements at a particular index
numbers[0]  # => 2
numbers[-1]  # => 11

# You can also slice lists - the same rules apply
letters[:3]  # => ['a', 'b', 'c']
numbers[1:-1]  # => [3, 5, 7]
```

Nested Lists

```
# Lists really can contain anything - even other lists!
combo = [letters, numbers]
combo  # => [['a', 'b', 'c', 'd'], [2, 3, 5, 7, 11]]
combo[0]  # => ['a', 'b', 'c', 'd']
combo[0][1]  # => 'b'
combo[1][2:]  # => [5, 7, 11]
```

General Queries

```
# Length (len)
len([])  # => 0
len("python")  # => 6
len([4, 5, "seconds"])  # => 3


# Membership (in)
0 in []  # => False
'y' in "python"  # => True
"minutes" in [4, 5, "seconds"]  # => False
```

Console I/O

Console I/O

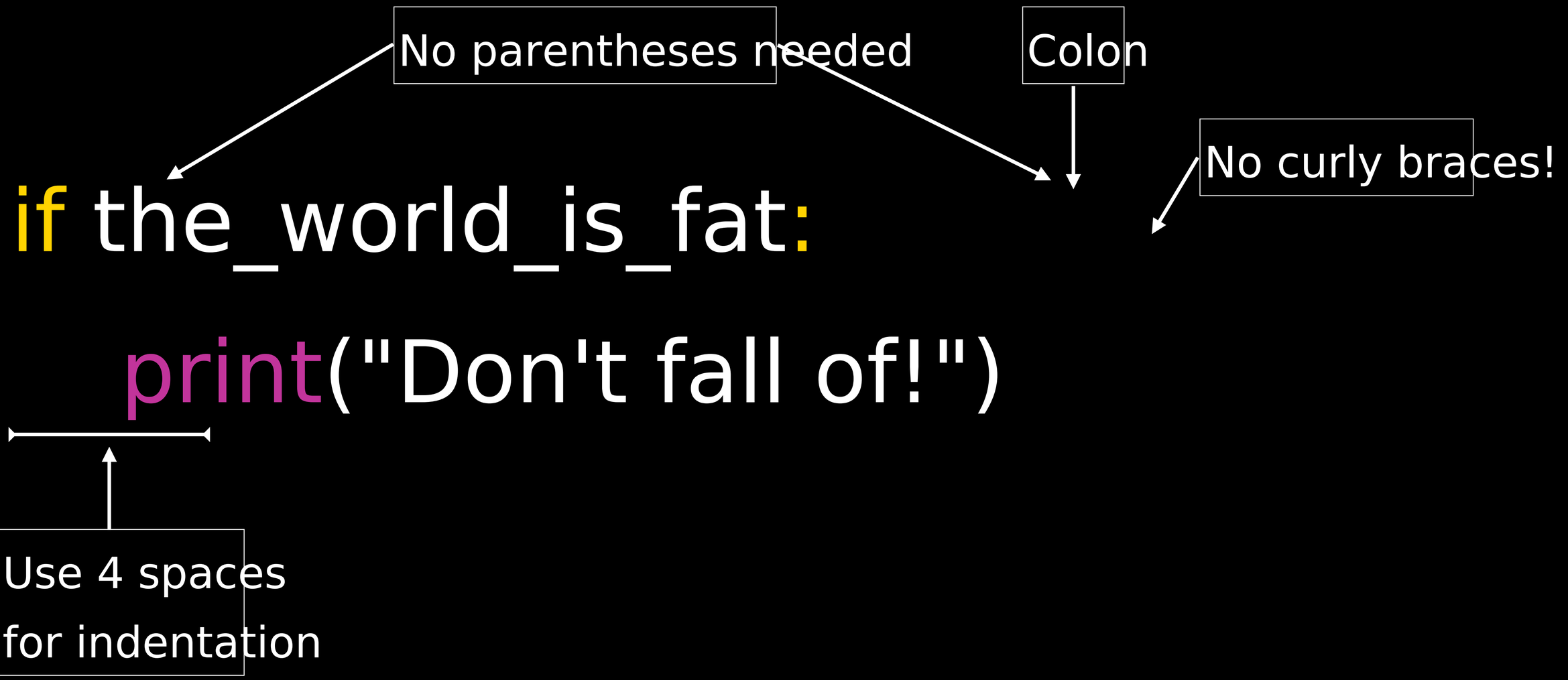
```
# Read a string from the user
>>> name = input("What is your name? ")
 What is your name?  Sam

>>> print("I'm Python. Nice to meet you,", name)
I'm Python. Nice to meet you, Sam
```

input prompts the user for inpu

print can be used in many diferent way

Control Flow

If Statements



4 Spaces?! No Braces?!

Zen of Python

Readability counts

Can be confgured in most development environments

elif and else

```
if some_condition:
    print("Some condition holds")
elif other_condition:
    print("Other condition holds")
else:
    print("Neither condition holds")
```

zero or more elifs

else is optional

Python has no switch statement, opting for if/elif/else chains

Palindrome?

Spelled the same backwards and forwards

```
# Is a user-submitted word a palindrome?
word = input("Please enter a word: ")
reversed_word = word[::-1]
if word == reversed_word:
    print("Hooray! You entered a palindrome")
else:
    print("You did not enter a palindrome")
```

Pause: How did this work agai

Truthy and Falsy

```
# 'Falsy' values
bool(None)   # => False
bool(False)  # => False
bool(0)      # => False
bool(0.0)    # => False
bool('')     # => False

# Empty data structures are 'falsy'
bool([])  # => False

# Everything else is 'truthy'
bool(41)           # => True
bool('abc')        # => True
bool([1, 'a', []])  # => True

bool([False])  # => True
bool(int)      # => True
```

Checking for Truthiness

with Steven Colbert

```
# How should we check for an empty list?
data = []
...
if data:
    process(data)
else:
    print("There's no data!")
```

You should almost never te
if expr == True

Loops

For Loops

Loop explicitly over data

Strings, lists, etc.

```
for item in iterable:
    process(item)
```

No loop counter!

Looping over Strings and Lists

```
# Loop over characters in a string.
for ch in "CS41":
    print(ch)
# Prints C, S, 4, and 1


# Loop over elements of a list.
for number in [3, 1, 4, 1, 5]:
    print(number ** 2, end='|')
# => 9|1|16|1|25|
```

Compare

```
String s = "CS41";
for (int i = 0; i < s.length(); ++i) {
    char ch = s.charAt(i);
    System.out.println(ch);
}
```

range

Iterate over a sequence of numbers

```
range(3)
# generates 0, 1, 2

range(5, 10)
# generates 5, 6, 7, 8, 9

range(2, 12, 3)
# generates 2, 5, 8, 11

range(-7, -30, -5)
# generates -7, -12, -17, -22, -27
```

range(stop) or range(start, stop[, step])

break and continue

```
for n in range(2, 10):
    if n == 6:
        break
    print(n, end=', ')
# => 2, 3, 4, 5,
```

break breaks out of the smallest enclosing for or while loop

```
for letter in "STELLAR":
    if letter in "LE":
        continue
    print(letter, end='*')
# => S*T*A*R*
```

continue continues with the next iteration of the loo

while loops

No additional surprises he

```
# Print powers of three below 10000
n = 1
while n < 10000:
    print(n)
    n *= 3
```

Functions

Dive into Python functions Week 3

Writing Functions

The def keyword defnes a function

Parameters have no explicit types

```
def fn_name(param1, param2):
    value = do_something()
    return value
```

return is optional

if either return or its value are omitted, implicitly returns None

Prime Number Generat

Prime Number Generator

```
def is_prime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

n = int(input("Enter a number: "))
for x in range(2, n):
    if is_prime(x):
        print(x, "is prime")
    else:
        print(x, "is not prime")
```

More to See

Keyword Arguments

Variadic Argument Lists

Default Argument Values

Unpacking Arguments

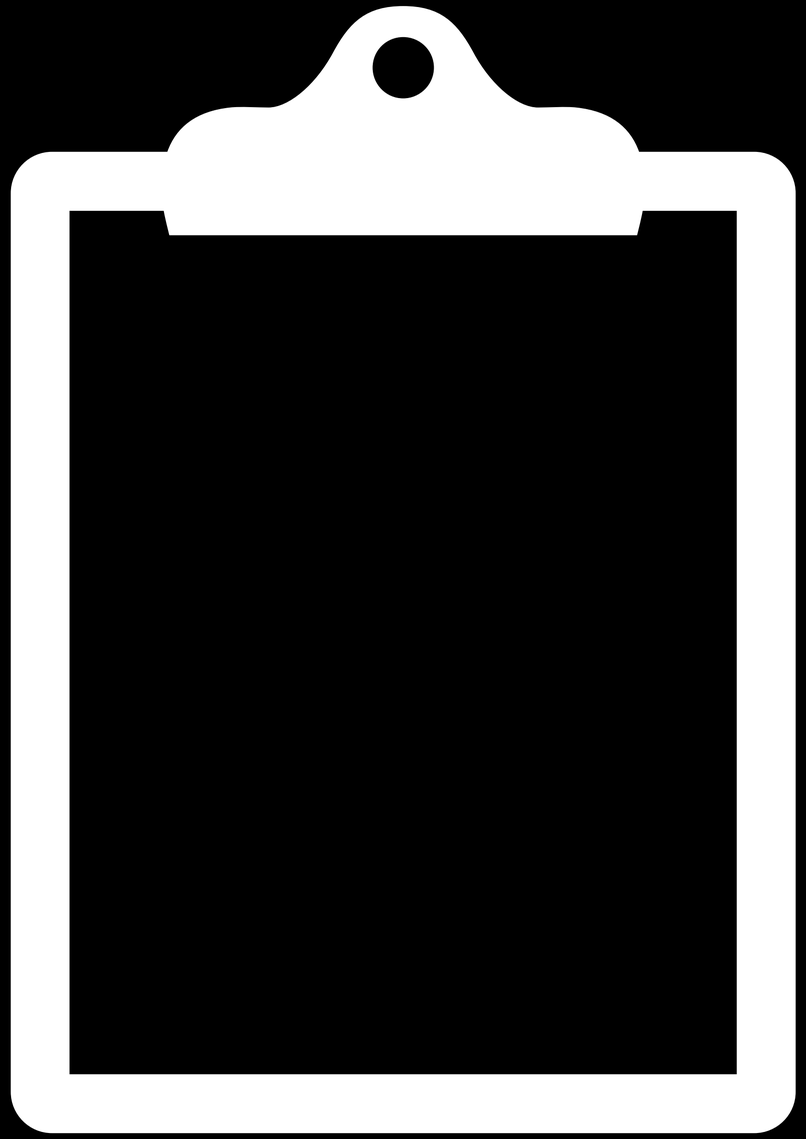
Anonymous Functions

First-Class Functions

Functional Programming

Next Time

More Python Fundamentals!



Types and Objects

String Formatting

File I/O

Using Scripts

Confguring Python 3

Lab!

Appendix

Citations

Examples in slides and interactive activities in thi

are drawn, with or without modifcation, from:

http://learnpythonthehardway.org/

http://learnxinyminutes.com/docs/python3/

https://docs.python.org/3/tutorial/index.html