

Welcome to Python!

CS 41: Hap.py Code
The Python Programming Language

Welcome to CS 41!

Today in CS 41

- Welcome!
- Who are we?
- Who are you?
- Why take CS 41?
- What is Python?
- Course Logistics
- Python Crash Course



Who are we?

INSTRUCTORS

Parth Sarin
psarin@stanford.edu



Michael Cooper
coopermj@stanford.edu



Who are we?

COURSE STAFF

Antonio Ferris



Zheng Lian



Sam Kwong



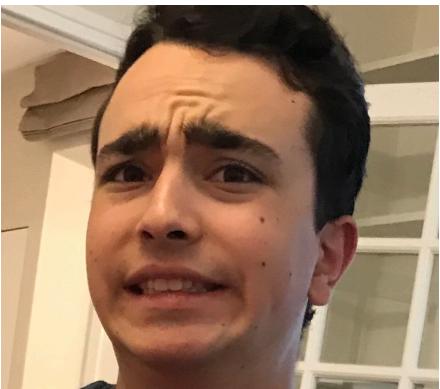
Andrew Huang



Alexander Iyabor Jr



Theo Culhane



Joy Hsu



Nick Negrete



Lemuel Cardenas-Arriaga



Our thanks to...

CONTRIBUTORS THROUGH THE YEARS

Sam Redmond



Unicorn



Colin Norick



Who are you?

Mechanical Engineering

Earth Systems

History

Civil and Environmental Engineering

Mathematical and Computational Science

Economics

Sociophonetics

Systems Engineering

Computer Science

Biophysics

International Relations

Electrical Engineering

Management Science and Engineering

Bioengineering

Medicine

Linguistics

Symbolic Systems

Science and Technology for Society

Political Economics

Science and Technology in Society

Computational Biology

Why CS 41?

Course Goals

1. Python fundamentals
2. Recognize and write “good” Python
3. Use Python for practical tasks...
4. ...especially machine learning and data science
5. Understand Python’s strengths (and weaknesses)

Your Questions

What is Python?

Will Python help me get a job / help me in the real world?

What is Python, anyway?



Guido Van Rossum
Former Benevolent Dictator for Life (BDFL)

- Python 1: 1994
A middle ground between C and shell scripts.
- Python 2: 2000
- Python 3: 2008
We're using Python 3.8

Your Questions

Will Python help me get a job / help me in the real world?

Python at Stanford

ALP 301: Data-Driven Impact

BIOS 203: Introduction to Atomistic Simulations for Biochemical Applications

BIOS 274: Introductory Python Programming for Genomics

CEE 345: Network Analysis for Urban Systems

COMM 277P: Programming in Journalism

CS 273C: Cloud Computing for Biology and Healthcare

CSRE 184E: Race, Gender, and Literary Digital Humanities

ENGLISH 184D: Race, Gender, and Literary Digital Humanities

GENE 222: Cloud Computing for Biology and Healthcare

MS&E 448: Big Financial Data and Algorithmic Trading

PHYSICS 368: Computational Cosmology and Astrophysics

PSYCH 162: Brain Networks

PSYCH 249: Large-Scale Neural Network Modeling for Neuroscience

PSYCH 253: High-Dimensional Methods for Behavioral and Neural Data

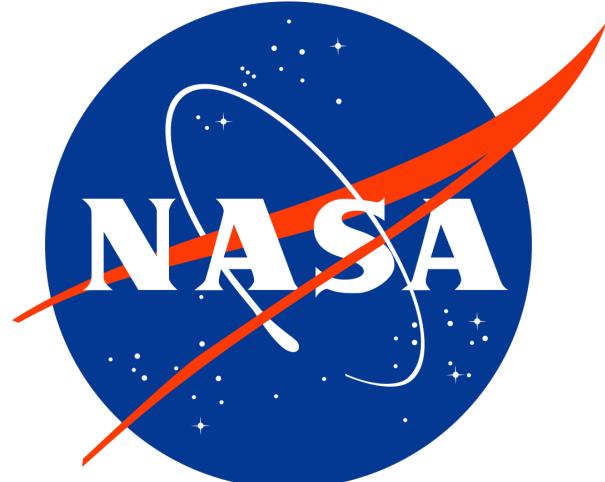
Your Questions

Will Python help me get a job / help me in the real world?

Python in Government



Smithsonian



Your Questions

Will Python help me get a job / help me in the real world?

Python in Business



Mount & Blade



Python is opinionated. This is its philosophy.

```
>>> import this
```

```
>>> 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 first 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

Case Study: Hello World

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

```
$ javac HelloWorld.java
$ java HelloWorld
Hello world!
```

Case Study: Hello World

```
// C++
#include <iostream>
using namespace std;

int main() {
    cout << "Hello world!" << endl;
    return 0;
}
```

```
$ g++ helloWorld.cpp
$ ./a.out
Hello world!
```

Case Study: Hello World

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

```
$ python helloworld.py  
Hello world!
```

Course Logistics

Logistics

Schedule	Monday & Wednesday, 3pm to 4:20pm Gates B12
Units	2 CR/NC
Website	https://stanfordpython.com
Prereqs	CS 106B/X or equivalent
Enrollment	Capped at 40 Auditing encouraged :)
Attendance	Mandatory (2 unexcused absences)
Piazza	Sign up!

Logistics

Assignments	4 total (3 assignments + final project)
Grading	Checkmark scale, functionality & style You'll get credit if you average a check
Late Days	Two 24-hour extensions
Honor Code	Don't cheat
OAE	See us for OAE accommodations.
Office Hours	Michael: Tues, 9:30am-10:30am, Gates B02 Parth: Fri, 3pm-4pm, Gates B02
Mascot	

On Unicorns...

I like unicorns... a lot.

I (currently) have ~10 stuffed unicorns and one real one.

There will be a lot of unicorns in this class.

Pro tips:



Unicorn
(I have this as a tote bag)



Not a Unicorn
(Wings)



Not a Unicorn
(Horses aren't even related!)



Acceptable
(and adorable)

The Road Ahead...



The Road Ahead

- Week 1:** Welcome & Basics
- Week 2:** Data Structures
- Week 3:** Functions and Files
- Week 4:** Functional Programming
- Week 5:** Python & the Web



The Road Ahead

Week 6: NumPy

Week 7: Standard Library &
Third Party Tools

Week 8: Plotting

Week 9: Machine Learning

Week 10: Advanced Topics

Weeks 8, 9, and 10 will
be more focused on
machine learning



Action Items

[Sign up for Piazza!](#)

Get your enrollment code!

<https://stanfordpython.com/admissions.html>

Assignment 0 has been released!

Set up Python

Experiment with Python Basics

You'll have the background to do this after Wednesday

On the horizon: Michael will send out a survey about your favorite movies.

We'll implement the Netflix Recommendation Algorithm with this data in Week 6's Lab! ;)

Today's Content

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

Interactive Interpreter

Interactive Interpreter



michaelcooper — Python — 80×24

```
Last login: Sat Dec 28 22:26:01 on ttys002
[Michael-Coopers-MacBook-Pro:~ michaelcooper$ python3
Python 3.8.0 (v3.8.0:fa919fdf25, Oct 14 2019, 10:23:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

It's Up to Interpretation...

- Python is an *interpreted language*
 - Means no compiler. The *Python Interpreter* parses the .py file and interprets the code as it goes.
 - There is no prior compilation of source code into machine code.
- Features and *interactive interpreter* (what we just saw)
 - Sandbox to experiment with Python.
 - Shortens code-test-debug cycle.
 - This is the greatest thing in the world.
 - Assignment 0 will give you experience setting up – and using – the interactive interpreter!

Comments

Some Brief Commentary...

```
# A single-line comment in Python is denoted with the hash symbol.
```

```
"""
Multi-line comments
Lie between quotation marks
This is a haiku
"""
```

Variables and Types

Variables

```
int x = 5;      # C/C++/Java  
  
x = 5          # Python - no type! No semicolon! What just happened?
```

A Type Dream

- Variables in Python are *dynamically typed*. They take on the type of the object they are representing.

```
>>> x = 1
>>> type(x)
<class 'int'>
>>> x = "Unicorn"
>>> type(x)
<class 'str'>
```

Numbers and Booleans

The Math of Least Resistance

- Python has only three numeric types: float, int, and complex.

```
5          # => 5      (int)
5.0        # => 5.0    (float)

1 + 8      # => 9      (int)
8 + 5.1    # => 13.1   (float)
2 * 4      # => 8      (int)
2 / 3      # => 0.666666 (float)

7 // 3     # => 2      (int; integer division)
7 % 3      # => 1      (int; modulo operator)
3 ** 3     # => 27     (int; exponential operator)
```

Mathematical Assignment

- Appending an `=` to any of the prior operations allows for assignment based on the prior value of the variable.

```
x += 5      # => x = x + 5
x -= 3      # => x = x - 3

x *= 2      # => x = x * 2
x /= 2      # => x = x / 2

x //= 4     # => x = x // 4
x %= 3      # => x = x % 3
x **= 3     # => x = x ** 3
```

Veritas vos Liberabit

- The `bool` type is a subtype of `int`: `False == 0`, `True == 1`

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

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

5 == 5                    # => True
1 != 100                  # => True
3 > 6                     # => False
4 >= 9                    # => False
3 > 2 > 1                 # => True (3 > 2 and 2 > 1)
```

Strings and Lists

Strings

- Python doesn't have an explicit `char` type. Instead, chars are strings with length 1.
- Both `" ... "` and `' ... '` define string literals.
- Python strings are Unicode by default (yes – this means we can use emoji in Python strings! 😊 ❤️ 🚀 👍)

```
>>> x = "Parth loves 🦄"  
>>> x + "!"  
"Parth loves 🦄!"  
# String concatenation using "+"
```

- You can only concatenate strings with strings! If `x` is an integer, it must be cast as a string (`str(x)`) before being concatenated with another string.

Indexing Strings

```
course = "happy code"
```

The diagram illustrates the indexing of the string 'course'. A horizontal line represents the string, divided into 12 segments by vertical grid lines. The first segment is index 0, and the last segment is index 11. The string itself is 'course = "happy code"'. The word 'happy' is highlighted in blue, and the word 'code' is also highlighted in blue. Below the string, three arrows point from labels to specific parts of the string: one arrow points to the start of 'happy' with the label 'Inclusive', another arrow points to the end of 'code' with the label 'Not Inclusive', and a third arrow points to the colon in 'stop:step' with the label 'Step'.

0	1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	---	----	----

course = "happy code"

Inclusive

Not Inclusive

Step

```
course[start:stop:step]
```

Inclusive

Not Inclusive

Indexing Strings

```
>>> course = "hap.py code"
>>> course[2]                      # Index at individual elements
'p'
>>> course[:3]                     # Index from beginning to element (exclusive)
'hap'
>>> course[5:]                      # Index from element to end (inclusive)
'y code'
```

Indexing Strings

```
course = "happy code"
```

The diagram illustrates string indexing for the variable `course`. The string value is "happy code". Above the string, indices from 0 to 11 are shown, and below it, indices from -11 to 0 are shown. The letters of the string are aligned with these indices. The word "happy" spans indices 0 to 5, and "code" spans indices 6 to 11. Indices 0 and 11 are at the start and end of the string, respectively.

```
course[start:stop:step]
```

Inclusive

Not Inclusive

Indexing Strings

```
>>> course = "hap.py code"
>>> course[2]                      # Index at individual elements
'p'
>>> course[:3]                     # Index from beginning to element (exclusive)
'hap'
>>> course[5:]                     # Index from element to end (inclusive)
'y code'
>>> course[-2]                     # Negative indexing indexes back from the end
'd'
>>> course[1:8:2]                  # Third parameter indicates a step size
'a.yc'
>>> course[8:1:-2]                 # Can do negative step sizes too
'o pp'
>>> # Your turn! How would we reverse the string using this notation?
>>> course[::-1]
```

Lists – Python’s ArrayList/Vector

```
# Create a new list
empty = []
letters = ["a", "b", "c"]
numbers = [1, 2, 3]

# Lists can contain elements of different types (even other lists!)
many_types = ["a", 2, 3, [4, 5]]

# Quick - what is many_types[3]?
# [4, 5]
```

Lists – Python’s ArrayList/Vector

```
# Create a new list
empty = []
letters = ["a", "b", "c"]
numbers = [1, 2, 3]

# Lists can contain elements of different types (even other lists!)
many_types = ["a", 2, 3, [4, 5]]

# Follow up - what is many_types[3][1]?
# 4
```

Lists – Python’s ArrayList/Vector

```
# Create a new list
empty = []
letters = ["a", "b", "c"]
numbers = [1, 2, 3]

# Lists can contain elements of different types (even other lists!)
many_types = ["a", 2, 3, [4, 5]]

# Use the extend function to append elements of a list to another list
>>> numbers.extend([4, 5, 6])
>>> numbers
[1, 2, 3, 4]
# Use the extend function to append elements of a list to another list
>>> numbers.extend([5, 6, 7])
>>> numbers
[1, 2, 3, 4, 5, 6, 7]
```

List Indexing and Slicing

```
# Create a new list
empty = []
letters = ["a", "b", "c"]
numbers = [1, 2, 3, 4]

# Indexing follows the exact same rules as string indexing
>>> letters[1]
'b'
>>> numbers[1:-1]
[2, 3]
>>> numbers[::-2]
[1, 3]
>>> numbers[1:3:-1]
[]                                     # What just happened here?
>>>
```

Queries Over Collections

```
# Length
>>> len([])
0
>>> len("CS41")
4
>>> len([1, 2, 3, "Parth"])
4

# Membership in a list
>>> 0 in [2, 3, 4]
False
>>> "5" in [3, 4, 5, "CS41"]
False                                # The integer 5 is a member, not the string!
# Checking for substrings using in keyword
>>> "tho" in "Python"
True
```

Console I/O

Console I/O

Read a string as user input and save it to a variable.

```
>>> fav_animal = input("What is your favourite animal? ")  
What is your favourite animal? Unicorn
```

```
>>> print("I also love " + fav_animal + "s!")  
I also love Unicorns!
```

Control Flow

Control Flow

```
if the_world_is_friendly:  
    print("Hello friendly world!")
```

- No parentheses necessary around the condition
- Colon at the end (no braces!)
- Use four spaces (or one tab) for indentation

elif and else

```
if the_world_is_friendly:  
    print("Hello friendly world!")  
  
elif the_world_is_happy:  
    print("Hello happy world!")  
  
else:  
    print("The world is neither friendly nor happy! It just needs a hug.")
```

- Both `elif` and `else` are optional.
- Python has no `switch` statement, instead preferring `if/elif/else` **chains**.

Veritas vos Liberabit II

```
# All objects in Python are either "truthy" or "falsy", meaning when cast  
# as booleans, they evaluate to either True or False.
```

```
# Zero values and empty data structures are "falsy"
```

```
>>> bool(None)                      # => False  
>>> bool(False)                    # => False  
>>> bool(0)                        # => False  
>>> bool(0.0)                      # => False  
>>> bool("")                        # => False  
>>> bool([])                       # => False
```

```
# Everything else is "truthy"!
```

```
>>> bool(42)                        # => True  
>>> bool("Sam Redmond")           # => True  
>>> bool([1, 2, 3])                # => True
```

```
# Quiz - is the following "truthy" or "falsy"?
```

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

Checking for Emptiness

```
# Truthiness and falsiness can be useful in checking whether or not a
# container is empty!

unicorns = []

if unicorns:
    frolic_with(unicorns)

else:
    print("No unicorns! 🙁")
```

Loops

Loops

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

- No loop counter! Instead, the `item` variable takes on the value of sequential elements of the iterable.
- Iterables can be strings, lists, or one of any one of a number of other data types (which we will see more of on Wednesday!).

Looping Over Strings and Lists

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

>>> for num in [3, 1, 4, 1, 5]:
    print(num**2)
9
1
16
1
25
>>>
```

Range

- The `range` function generates an iterable over a range of numbers.
- Syntax: `range(stop)` or `range(start, stop[, step])`
 - Same rules as string slicing! start is inclusive, stop is exclusive.

```
>>> 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
```

Break and Continue

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

for letter in "UNICORN":
    if letter in "UN":
        continue
    print(letter)
I
C
O
R
```

*# Break breaks out of the smallest enclosing
for or while loop*

*# Continue continues to the next iteration of
the loop*

Loops

- Syntax is very similar to the syntax for control flow.

```
while condition:  
    do_stuff()
```

While Loops - Example

```
# Print powers of three below 10000
>>> n = 1

>>> while n < 10000:
    print(n)
    n *= 3

1
3
9
27
81
243
729
2187
6561
```

Functions

Functions

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

- The `def` keyword defines a function.
- Parameters do not have explicit types (unlike C, C++, Java).
- `return` is optional. If either `return` or its value are omitted, it implicitly returns `None`.

Example – Is A Number Prime?

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

Assignment Expressions

Assignment Expressions

- New feature in Python 3.8!
- Use `:=` to assign variables from an expression!

```
# Example: ordering food at a restaurant
```

```
# The old way of doing things...
```

```
order = input("What would you like to order?")
while order != "Nothing else":
    print("You're ordering: " + order)
    order = input("What would you like to order?")
```

```
# Using assignment expressions!
```

```
while (order := input("What would you like to order?")) != "Nothing else":
    print("You're ordering: " + order)
```

- Remember – readability counts! Only use this if it doesn't compromise readability. 😊

Review

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

