

# Python Review Session

CS109 Autumn 2019



# Today's questions

How does Python compare to other programming languages?

How can I use Python to solve problems in CS109?

# Today's topics

1. Python vs. other languages
2. Features of Python
3. Python in action!

How does Python compare to  
other programming  
languages?

## Other programming languages: Java

```
ArrayList<Integer> evens = new ArrayList<Integer>();  
for(int i = 0; i < 100; i++) {  
    if(i % 2 == 0) {  
        evens.add(i);  
    }  
}  
System.out.println(evens);
```

## Other programming languages: C++

```
using namespace std;
vector<int> evens;
for(int i = 0; i < 100; i++) {
    if(i % 2 == 0) {
        evens.push_back(i);
    }
}
cout << evens << endl;
```

## Other programming languages: JavaScript

```
var evens = [];  
for(var i = 0; i < 100; i++) {  
    if(i % 2 === 0) {  
        evens.push(i);  
    }  
}  
console.log(evens);
```

# Python

```
evens = []  
for i in range(100):  
    if i % 2 == 0:  
        evens.append(i)  
print(evens)
```



# Python

```
evens = []  
for i in range(100):  
    if i % 2 == 0:  
        evens.append(i)  
print(evens)
```

```
# With a list comprehension instead  
print([i for i in range(100) if i % 2 == 0])
```

# Python Basics

*(adapted from Sam Redmond's CS41)*

# Python Interpreter

- If you have Python 3 installed:
  - Type **python3** on the command line
    - Instantly in the interactive interpreter!
      - Sandbox for experimenting with Python
      - Great for learning about how Python works!

# Comments

```
# single line comments use a `#`
```

```
# what the cool kids use these days #relevant
```

```
"""
```

```
Multiline comments written in between a pair of three
```

```
"s
```

```
"""
```

# Variables

## Java

vs.

## Python

```
int x = 0;
```

```
x = 0
```

- semicolons
- statically-typed

- no semicolons
- dynamically-typed
  - not declared with explicit type
  - but every object still has a type (so Python knows, even if you don't!)

# Numbers and Math

```
# Python has 2 numeric types: ints and floats
```

```
3          # 3 (int)
3.0        # 3.0 (float)
5 / 2      # 2.5 (float division)
5 // 2     # 2 (integer division)
5 % 2      # 1 (integer modulus)
5 ** 2     # 25 (exponentiation)
```

# Boolean + Other Operators

```
not          # instead of !
or           # instead of ||
and          # instead of &&
==           # like ==
is           # same Python object (same address)
```

# Booleans + Operators

# note that True and False start with capital letters

```
2 * 3 == 6          # True
not True            # False
True and False     # False
True or False      # True
3 < 2              # False
1 < 2 < 3          # True (1 < 2 and 2 < 3)
```



# Strings

```
# no char in Python
```

```
# ' ' and "" both create string literals
```

```
greeting = 'Hello'
```

```
audience = "world"
```

```
greeting + " " + audience + "!" # 'Hello world!'
```

# Indexing

s = 'Arthur'

A diagram illustrating string indexing for the string 'Arthur'. The string is enclosed in single quotes. Above each character, an index number is shown: 0 for 'A', 1 for 'r', 2 for 't', 3 for 'h', 4 for 'u', and 6 for the final 'r'. Vertical dashed lines connect each character to its corresponding index number. The string ends with a null terminator character, represented by a gray square.

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

# Negative Indexing

`s = 'Arthur'`

The diagram shows the string 'Arthur' with its characters aligned to indices. Above the string, indices 0 through 6 are shown in grey. Below the string, indices -6 through 0 are shown in white. Vertical dashed orange lines connect each character to its corresponding index: 'A' at 0 and -6, 'r' at 1 and -5, 't' at 2 and -4, 'h' at 3 and -3, 'u' at 4 and -2, and 'r' at 5 and -1. The final character 'r' is at index 6 and -0.

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

# Slicing

s = 'Arthur'

# Slicing

s = 'Arthur'

0 1 2 3 4 5 6

# Slicing

s = 'Arthur'

The diagram shows the string 'Arthur' with indices 0 through 6 above each character. Vertical dashed lines mark the start and end of each character. A white bracket is drawn below the characters 'A' and 'r', spanning from index 0 to index 2.

```
s[0:2] == 'Ar'
```

# Slicing

s = 'Arthur'

The diagram shows the string 'Arthur' with indices 0 through 6 above each character. Vertical dashed lines mark the boundaries of each character. Two horizontal brackets below the string indicate slices: one from index 0 to 2 (covering 'Ar') and another from index 3 to 6 (covering 'thur').

```
s[0:2] == 'Ar'
```

# Slicing

`s = 'Arthur'`

The diagram shows the string 'Arthur' with indices 0 through 6 above each character. Vertical dashed lines separate the characters. Two horizontal brackets below the string indicate slices: one from index 0 to 2 (covering 'Ar') and another from index 3 to 6 (covering 'hur').

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



# Slicing

`s = 'Arthur'`

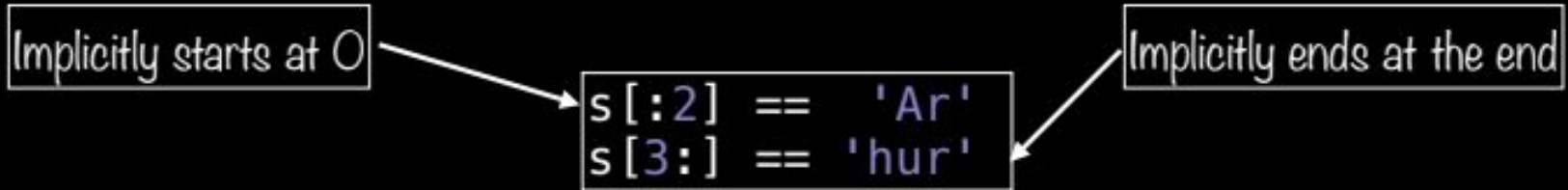
The diagram shows the string 'Arthur' with indices 0 through 6 above each character. Vertical dashed lines separate the characters. Three horizontal brackets below the string indicate slicing operations: one from index 0 to 2 (under 'Ar'), one from index 3 to 6 (under 'hur'), and one from index 1 to 4 (under 'rth').

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

# Strings

`s = 'Arthur'`

0 1 2 3 4 5 6

A diagram illustrating string indexing for the string 'Arthur'. The string is enclosed in single quotes. Above each character, its corresponding index is shown: 0 for 'A', 1 for 'r', 2 for 't', 3 for 'h', 4 for 'u', 5 for 'r', and 6 for the closing quote. Vertical dashed lines connect each character to its index.

# Strings

`s = 'Arthur'`

0 1 2 3 4 5 6

The diagram shows the string 'Arthur' with indices 0 through 6 above each character. Vertical dashed lines connect the indices to the characters. A horizontal double-headed arrow is drawn below the first three characters, 'Ar', indicating a slice from index 0 to 2.

Implicitly starts at 0

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

Implicitly ends at the end

# Strings

`s = 'Arthur'`

0 1 2 3 4 5 6

The diagram shows the string 'Arthur' with indices 0 through 6 above each character. Vertical dashed lines connect the indices to the characters. Two horizontal brackets are shown below the string: one from index 0 to 2, and another from index 3 to 6.

Implicitly starts at 0

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

Implicitly ends at the end

# Strings

`s = 'Arthur'`

0 1 2 3 4 5 6

One way to  
reverse a string

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

Can also pass a step size

# Lists

```
[1, 2, 3, 4, 5]
```

```
['a', 'b', 'b', 'd']
```

```
[True]
```

```
[1, 'a', 2, 'b', True]
```

```
[] # empty list
```

## Definition

### List

A data type for storing values in a linear collection.

*Similar to ArrayList/Vector  
Common and versatile*

# Inspecting list elements

```
>>> letters = ['a', 'b', 'c', 'd']
```

```
>>> letters[0]
```

```
'a'
```

```
>>> letters[1:]
```

```
['b', 'c', 'd']
```

## Len() built-in

```
>>> letters = ['a', 'b', 'c', 'd']
```

```
>>> len(letters)
```

4



## Adding elements

```
>>> lst = [1, 2, 3, 4, 5]
```

```
>>> lst.append(6)
```

```
>>> lst
```

```
[1, 2, 3, 4, 5, 6]
```

```
>>> lst += [7, 8]
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```


## Removing elements

```
>>> lst = [1, 2, 3, 4, 5]
```

```
>>> last_elem = lst.pop()
```

```
>>> last_elem
```

5



*pop() removes the last element in a list and returns it. You can also pass an index into pop().*

## Membership queries

```
>>> fruits = ['apple', 'banana', 'mango', 'kiwi']
```

```
>>> 'mango' in fruits
```

**True**

```
>>> 'broccoli' in fruits
```

**False**

```
>>> 'broccoli' not in fruits
```

**True**

## For loops – foreach

```
>>> fruits = ['apple', 'banana', 'mango']
```

```
>>> for fruit in fruits:
```



*foreach loop*

```
...     print(fruit)
```

apple

banana

mango

## For loops – for (range)

```
>>> fruits = ['apple', 'banana', 'mango']
```

```
>>> for i in range(len(fruits)):
```

```
...     print(fruits[i])
```

apple

banana

mango

*for(int i = 0; i < fruits.length; i++)*



# Range

```
# range(start_i, end_i, step)
```

```
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, -11, -1) # generates -7, -8, -9, -10
```

## For loops — enumerate

```
>>> fruits = ['apple', 'banana', 'mango']
```

```
>>> for i, fruit in enumerate(fruits):
```

```
...     print(i, fruit)
```

```
0 apple
```

```
1 banana
```

```
2 mango
```

# Control flow

```
if cs109 == 'awesome':  
    print('I love this class!')
```

- No parentheses needed for boolean expression
- Colon
- No curly braces
- 4 spaces for indentation



# Control flow

```
if some_condition:
```

```
    print('Some condition holds')
```

```
elif other_condition:
```

```
    print('Other condition holds')
```

```
else:
```

```
    print('Neither condition holds')
```

```
# No switch statement in Python!
```

# Writing functions

```
def function_name(param1, param2):  
    result = do_something()  
    return result
```

- **def** keyword defines a function
- no explicit types for parameters

# Functions can return multiple values!

```
def function_name(param1, param2):  
    result1 = do_something()  
    result2 = do_something_else()  
    return result1, result2
```

- Returns a tuple
  - tuples are an immutable type
  - can be packed/unpacked

## Other important topics we'll cover more in the demo:

- Getting set up with Python
  - using Jupyter notebooks
- Dictionaries
- Sets
- Tuples
- Other libraries
- File reading

## Other resources

CS 41 lectures/slides: <https://stanfordpython.com/>

Resource list from summer CS106AP:

<https://web.stanford.edu/class/cs106ap/handouts/additional-resources.html>

# Demo Time!

[https://github.com/sjohnsonyu/cs109\\_python\\_tutorial](https://github.com/sjohnsonyu/cs109_python_tutorial)