INTRO TO PYTHON
STRINGS AND CONDITIONALS

Shannon Turner

Twitter: @svthmc

GitHub: @shannonturner

OBJECTIVE

- Learn about variables
- Learn how to display and modify text
- Learn about conditionals
- Learn how to use conditionals to change how your program behaves

VARIABLES

- Variables are containers for information; you can store text, numbers, or any other type of thing!

```
twitter = "@hearmecode"
members = 3000
```

THE PRINT COMMAND

- Use the **`print`** command to show some information to the screen.

```
print "Welcome to Hear Me Code!"
print '3000 members and growing'
```

- Since we created a variable on the previous slide, we can use it now:

```
print twitter
```

THE PRINT COMMAND

- Let's take a closer look at the difference between these two:

```
print twitter
```

```
print "twitter"
```

STRINGS: IT'S JUST TEXT

- Strings are a way to store information
  - Addresses
  - Email addresses
  - URLs
  - Names (people, places, …)
  - Phone Numbers
  - so much more (anything with text!)

STRINGS: IT'S JUST TEXT

- Strings are combinations of characters
  - Letters
  - Numbers
  - Punctuation
  - Basically anything you can make on the keyboard and then some
  - Special characters, like tabs and newlines

USE QUOTES AROUND STRINGS

- How to spot a string: it has quotes around it

```
"This is a string"
'This is also a string'
```

- Using single or double quotes comes down to personal preference … as long as you start and end a string with the same quote

```
'Not like this :("
```

SINGLE OR DOUBLE QUOTES?

- If your text contains a single quote, you'll want to use double quotes around your text:

```
1  print "My governor's name is Martin O'Malley"
```

If you don't, you'll get an error:

```
1  print 'My governor's name is Martin O'Malley'
```

Try both of these out!

USE TRIPLE QUOTES FOR LONG STRINGS

- If you have a really long string, use three quote symbols in a row to start and end your string.

```
1 article = """At Hear Me Code, students are teachers in training.
2 The key to the classes' appeal, said Criqui, who is now an assistant
3 teacher at Hear Me Code? "It's by women, for women," she said..."""
```

SPECIAL CHARACTERS IN STRINGS

- Special Characters
  - **\n** Newline
  - **\t** Tab

```
>>> print "Contact Info:\n Shannon \t shannon@hearmecode.com"
Contact Info:
 Shannon         shannon@hearmecode.com
```


HEAR ME CODE

STRINGS: QUICK EXERCISE

- Print the following string:

```
Lesson          Topic
1               Strings and Conditionals
2               Lists and Loops
3               Dictionaries & Files
```

- Keep in mind you'll need to use tabs & newlines!

HOW LONG IS MY STRING?

- `twitter = "@hearmecode"`
- `print len(twitter)`
- `len()` works on lists, too! We'll work with **lists** in Lesson 2.

STRINGS: SLICING

- **`twitter = "@hearmecode"`**

- Slicing lets you see individual pieces or "slices" of your string*

*Slicing also works with **lists** in the same way.

STRINGS: SLICING

- **`twitter = "@hearmecode"`**
- **`print twitter[0]`**
  - Here, 0 refers to the **index** that you want to see
  - Slicing on first_name and last_name can give us a person's initials; slicing on phone number can give us the area code

0	1	2	3	4	5	6	7	8	9	10
@	h	e	a	r	m	e	c	o	d	e
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

STRINGS: SLICING

- You can get more than one item in a slice
  - **`twitter[1:5]`**
    - The **index** on the left (1) is where you start
    - The **index** on the right (5) is where you end, but Python **stops short** and doesn't include it

0	1	2	3	4	5	6	7	8	9	10
@	h	e	a	r	m	e	c	o	d	e
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

STRINGS: SLICING

- The indices you provide are optional!

`twitter[:5]`

The left index is not provided, so Python assumes you want to start at the beginning and stop just short of item 5

0	1	2	3	4	5	6	7	8	9	10
@	h	e	a	r	m	e	c	o	d	e
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

STRINGS: SLICING

- The indices you provide are optional!

`twitter[1:]`

The right index is not provided, so Python assumes you want to start at item 1 and go to the end

0	1	2	3	4	5	6	7	8	9	10
@	h	e	a	r	m	e	c	o	d	e
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

STRINGS: SLICING EXERCISE

- **`phone = "(202) 456-7890"`**
- Use slicing to print out the area code
- And then the middle three numbers
- All finished? Print out the last four digits

STRINGS: .FORMAT( )

```
1 name = "Shannon"
2 age = 29
3 print "My name is: {0} and my age is: {1} ".format(name, age)
```

- Think of the numbers as placeholders for your variables (or like Mad libs!)

- Remember, Python starts counting at zero. So zero is the first variable, which corresponds to **`name`**

STRINGS: .FORMAT( ) AND SLICING

```
1  phone = "202-555-6789"
2  print "Call {0} for great pizza".format(phone[4:])
```

- You can use slicing with **`.format()`**
- What does **`phone[4:]`** evaluate to?

STRINGS: .FORMAT( ), FUNCTIONS, AND MATH

```
1 tweet = """In just over one year, @hearmecode has reached over 800
2 members who are learning how to code together."""
3
4 print """That tweet is {0} characters.
5 You have {1} characters left""".format(len(tweet), 140-len(tweet))
```

- You can also do math inside **`.format()`**!
- And use functions!
- And so much more!

STRINGS: FORMATTING EXERCISE

- `phone = "202-555-9876"`

- Use `.format()` and slice the **phone** variable to print these:
  - **Area Code: 202**
  - **Local: 555-9876**
  - **Different format: (202) 555-9876**

STRINGS: STRING METHODS

- String methods let you perform special actions on your strings
  - Replace one part of a string with another
  - Find one part of a string within the string
  - Count the number of times one part of a string appears within the string
  - … and many more!

STRING METHOD: .FIND( )

```
>>> email = "shannon@hearmecode.com"
>>> print email.find("@")
7
```

- **`.find()`:** Like Ctrl+F in most programs

- The number you get back is the index (slice) where you found the item.

```
>>> print email.find("Z")
-1
```

STRING METHOD: .REPLACE( )

```
>>> twitter = "@hearmecode"
>>> print twitter.replace('@', '#')
#hearmecode
>>> print twitter
@hearmecode
```

- **`.replace()`:** Like Find+Replace in Word, Excel, etc.

  … wait a second! Why didn't it save the changes?

STRING METHOD: .REPLACE( )

- Change it just for now:

```
>>> twitter = "@hearmecode"
>>> print twitter.replace('@', '#')
#hearmecode
>>> print twitter
@hearmecode
```

- Making the changes stick:

```
>>> twitter = twitter.replace('@', '#')
>>> print twitter
#hearmecode
```

HOW FUNCTIONS WORK

- **Arguments/parameters** tell a **function** or **method** how to do their action, or what to do it to.

- **`len(tweet)`**

- Function (action): **`len()`**
- Argument/parameter: **`tweet`**

ARGUMENTS

- **Arguments/parameters** tell a **function** or **method** how to do their action.

- `twitter.replace("@", "#")`

- String method (action): `.replace()`
- Argument/parameter: `"@"` and `"#"`
- Where does Python perform the find/replace? On the string that comes before the dot!

RETURN VALUES

- Some functions and methods give you **return values** when they're finished so you know what happened.

- You can save this return value into a variable.

```
1  length = len(tweet)
2  # tweet: the string to measure
3  # len(): function that finds the length
4  # length: a new variable, the length is stored here
```

RETURN VALUES

- Some functions and methods give you **return values** when they're finished so you know what happened.

- You can save this return value into a variable.

```
 6  position = phone.find("(")
 7  # .find("("): look for a left parenthesis
 8  #              in the variable phone
 9  # phone: a string containing a phone number
10  # position: a new variable, the position is stored here
```

STRING METHOD: .STRIP( )

```
>>> address = "                    123 Sesame Street                "
>>> print address
                    123 Sesame Street
>>> print address.strip()
123 Sesame Street
```

`.strip()`

- Removes whitespace from the beginning and end of a string (not the middle)

STRING METHODS: .LOWER( ), .UPPER( )

.lower()

.upper()

```
>>> state = 'District of Columbia'
>>> print state
District of Columbia
>>> print state.upper()
DISTRICT OF COLUMBIA
>>> print state.lower()
district of columbia
```

- Converts a string to all lowercase or all uppercase

STRING METHOD: .COUNT( )

```
1 article = """At Hear Me Code, students are teachers in training.
2 The key to the classes' appeal, said Criqui, who is now an assistant
3 teacher at Hear Me Code? "It's by women, for women," she said..."""
4
5 print article.count(" he said")
6 print article.count(" she said")
```

`.count()`

- How many times was a woman quoted in this article?
- How many times was a man quoted?

CONDITIONALS

- We use conditionals to compare two things
- Conditionals allow us to change our program's behavior based on the results

CONDITIONALS

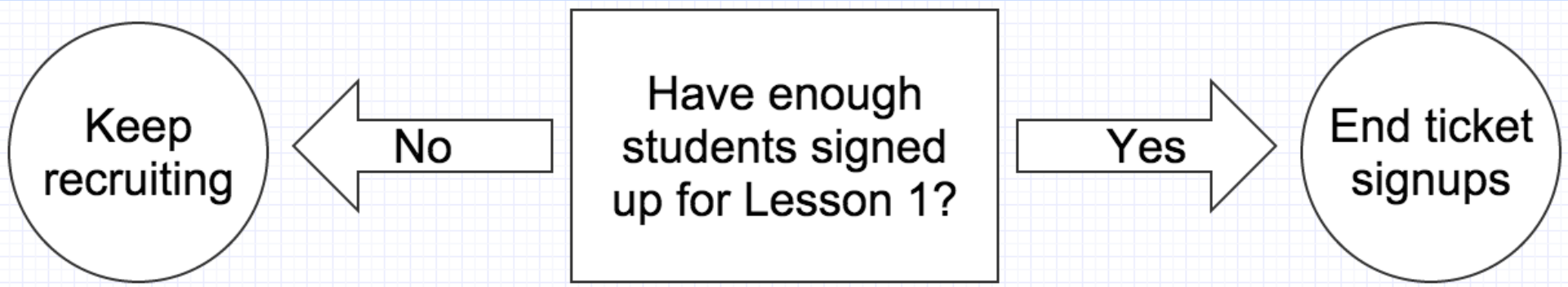
- Conditional: just a fancy name for a yes or no question
- Conditionals are ways to compare things and use that information to make decisions
- Conditionals can let you change the behavior of your program

CONDITIONALS

- Ways to think about conditionals

- Is this a valid email address?
- Does my phone number have enough digits?
- Are more people signed up for my event than the room can hold?

CONDITIONALS



CONDITIONALS

```
1 students = 10
2 capacity = 50
3
4 if students < capacity:
5     print "Keep recruiting"
6 else:
7     print "End ticket signups"
8
```

CONDITIONALS

- Conditionals are paired with if statements, which ask whether or not the conditional is true

```
4  if students < capacity:
5      print "Keep recruiting"
6  else:
7      print "End ticket signups"
```

- True or False: do we have more capacity than students?

CONDITIONALS

Operators (ways to compare two things)

`==` Equality operator (don't confuse with a single equals sign)

```
5 == 7 # Python says: False
5 == 5 # Python says: True
```

CONDITIONALS

Operators (ways to compare two things)

`>` Greater than operator

```
5 > 7 # Python says: False
5 > 2 # Python says: True
```

CONDITIONALS

Operators (ways to compare two things)

==	These **two** things are equal
!=	**NOT!** equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

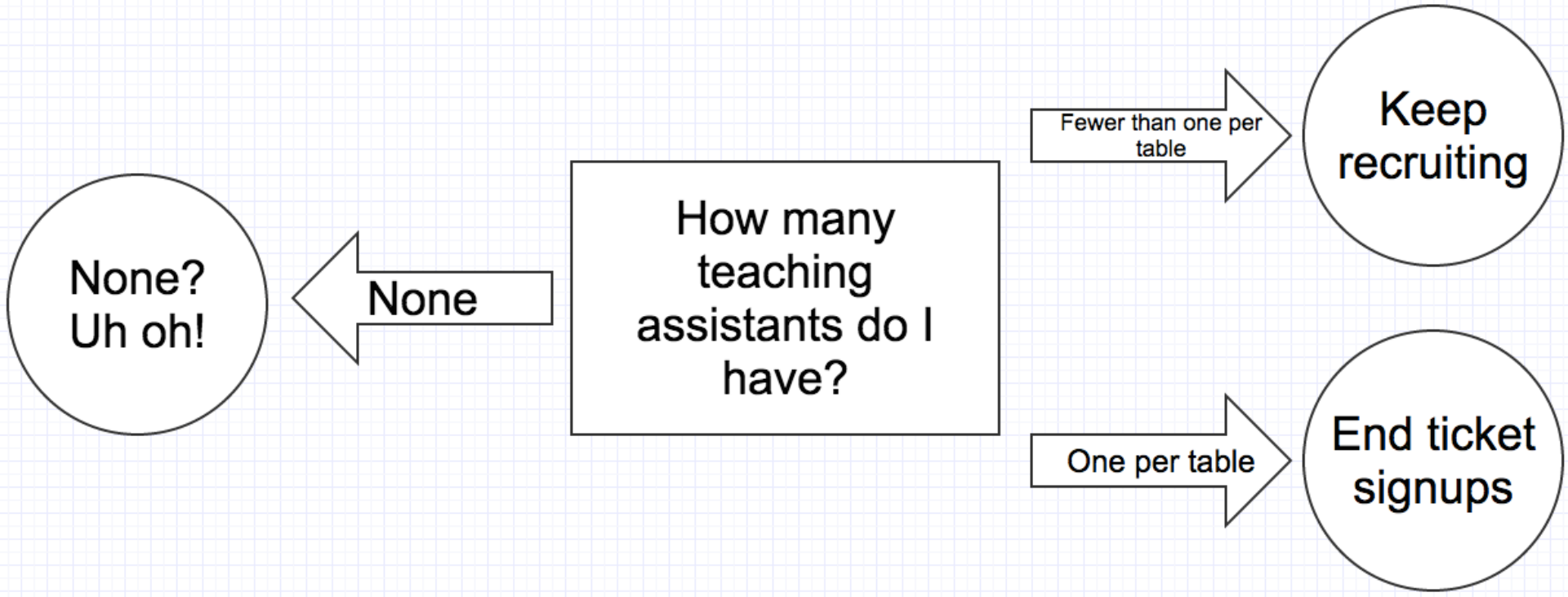
MULTIPLE CHOICE: ELIF

I live in

- ○ DC
- ○ Maryland
- ○ Virginia
- ○ None of the above

```
1  if location == 'DC':
2      print "District of Columbia"
3  elif location == 'MD':
4      print "Maryland"
5  elif location == 'VA':
6      print "Virginia"
7  else:
8      print "Somewhere else"
```

CONDITIONALS



CONDITIONALS

```
1  teaching_assistants = 5
2
3  if teaching_assistants == 0:
4      print "None? Uh oh!"
5  elif teaching_assistants < students / 5:
6      print "Keep recruiting TAs"
7  else:
8      print "Aren't the TAs great though?"
```

CONDITIONALS

```
1  students = 10
2  capacity = 50
3
4  teaching_assistants = 5
5
6  if students < capacity:
7      print "Keep recruiting"
8  else:
9      print "End ticket signups"
10
11 if teaching_assistants == 0:
12     print "None? Uh oh!"
13 elif teaching_assistants < students / 5:
14     print "Keep recruiting TAs"
15 else:
16     print "Aren't the TAs great though?"
17
```

MORE EXAMPLES OF CONDITIONALS

```
1  capacity = 50
2  rsvps = 40
3
4  tickets_remaining = capacity - rsvps
5
6  if tickets_remaining <= 0:
7      # There might be fewer than zero tickets remaining
8      #   if the event is oversold
9      print "This event is sold out!"
```

MORE EXAMPLES OF CONDITIONALS

```
1  # Name isn't blank
2  # Read this as "if name is not equal to nothing"
3  # (In other words, if it has any value at all)
4  # Notice that there is NOT a space between the quotes.
5  if name != '':
6      print "Hello, {0}!".format(name)
7  else:
8      # Name was left blank
9      print "Name cannot be blank!"
10
11 # Lines 5-9 are functionally equivalent to 13-16.
12 # It's just a small difference in coding style.
13 if name == '':
14     print "Name cannot be blank!"
15 else:
16     print "Hello, {0}!".format(name)
```

CONDITIONALS: EXERCISE

- Create two variables (**volunteers**, **goal**)
- Tell the user whether they are **above**, **below**, or **at** their recruitment goal.

- Example:

  Volunteers: 90

  Goal: 100

>>> You are behind!

USING THE IN KEYWORD

The **in** keyword, when used with a conditional, tells you whether some text appears **in** a string.

```
1  article = """Hear Me Code is more than just a coding class
2      ... it's just as much about building community
3      and developing the next set of women leaders in tech."""
4
5  if "Hear Me Code" in article:
6      print "Hear Me Code was mentioned in this article!"
7  else:
8      print "This article is about something else."
```

COMPLEX CONDITIONALS

You can use functions and methods as part of your conditional!

```
1  if state.upper() == 'DC':
2      print "District of Columbia"
3
4  if email_address.count('@') > 1:
5      print "{0} is not a valid email".format(email_address)
6
7  if len(phone) < 7:
8      print "This phone number is too short!"
```

COMPOUND CONDITIONALS

Using the **and** keyword, both conditions must be true for the print statement at line 7 to run.

```
1 article = ' ... '
2
3 men_quoted = article.count(' he said')
4 women_quoted = article.count(' she said')
5
6 if men_quoted == 0 and women_quoted == 0:
7     print "Neither men nor women were quoted"
```

COMPOUND CONDITIONALS

Using the **or** keyword, either condition could be true for the print statement at line 2 to run.

```
1  if state == 'DC' or state == 'MD' or state == 'VA':
2      print "Welcome to Hear Me Code!"
```

NESTED CONDITIONALS

```
1  if lesson == 1:
2      print "Strings and Conditionals"
3  elif lesson == 2:
4      print "Lists and Loops"
5  elif lesson == 3:
6      print "Dictionaries and Files"
7  elif lesson >= 4:
8      print "We don't teach these in person anymore"
9      print "But they're available for self-study"
10
11     if lesson == 4:
12         print "Functions"
13     elif lesson == 5:
14         print "APIs"
```

PLAYTIME!

Head to **Hear Me Code's Slides on Github**, which has examples & code samples for the lesson.

See the **playtime** folder for the playtime exercise for this lesson.