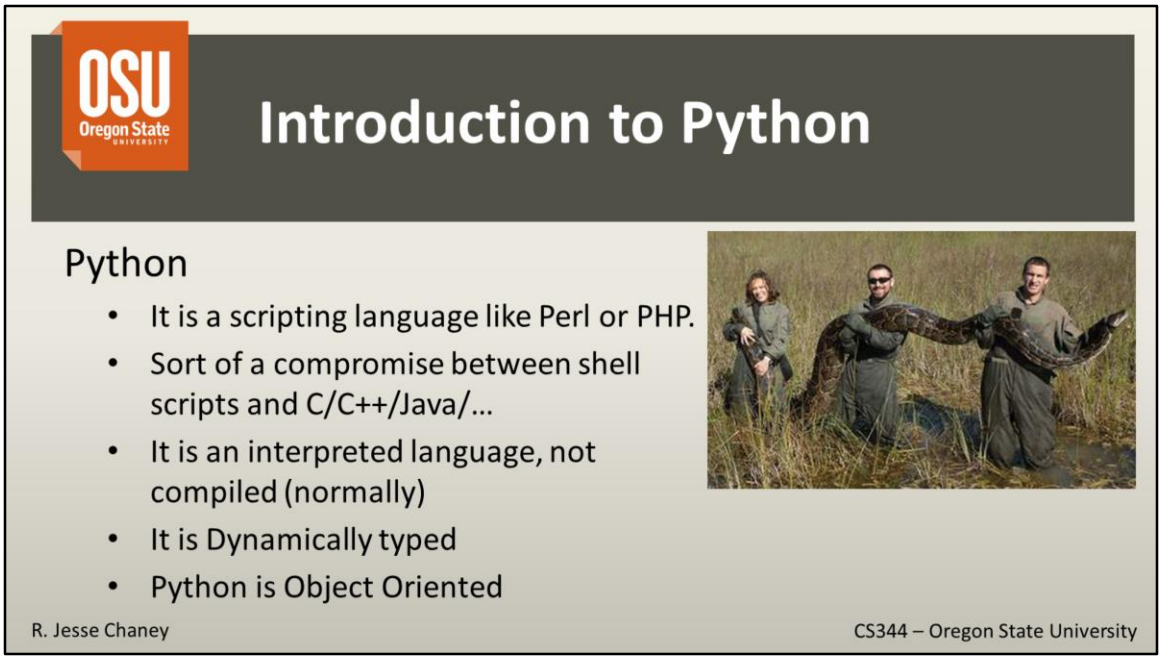


This class has a requirement that I teach a scripting language, so I'm going to teach some Python.

Python is a scripting language like Perl or PHP. More like Perl.

I avoided learning Python for several years for a couple reasons.

1. I already knew Perl very well and was an efficient Perl developer. Python did not seem to offer anything beyond what I could do in Perl. I'd written quite lot of Perl. I did know that Perl is sometimes considered a write-only language.
2. **Python uses whitespace to designate code blocks**. We go over this more in a minute, but I was worried that accidentally hitting a tab or space while reviewing my code could drastically alter the function of the code. Worse, what might happen when differing editors treated a TAB character differently.

Python is interpreted, not compiled. However, there are some Python compilers out there. One very common implementation, CPython, is both interpreted and compiled.

If you have not used a dynamically typed language before, you are probably going to

really like it.

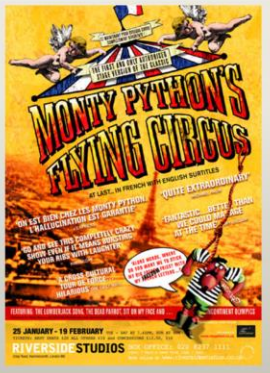
Python is object oriented, or at least can be.

For us, we are using the online documentation for Python. If you decide you really like Python, in addition to some excellent online resources (most of which are free), there are many MANY books about Python.

Python History

Python was developed by Guido van Rossum as a replacement for the ABC programming language. Development began in the late 1980's.

The newest development track is currently in Python 3, but we will be using Python 2.6 (the default on `os-class`).



Guido van Rossum developed the language and is still very active in its development. His title in the Python community is Benevolent Dictator for Life.

The language is named for the British comedy group Monty Python's Flying Circus.

Python 3 has actually been under development since 2008, but there are more utilities available with Python 2. I'm sure you'll be tempted by Python 3, but use Python 2 for this class. There are some differences that could cause you to loose points on assignments if you use Python 3. At least it is not Perl, where the "latest", Perl 6, has been under development since 2000 and does not have much support at all.

Python Major Users

The web development frameworks Django and Pyramid

Python is used in some scientific and GIS libraries because of how easily it handles floating point calculations.

Google uses it a lot.

R. Jesse Chaney

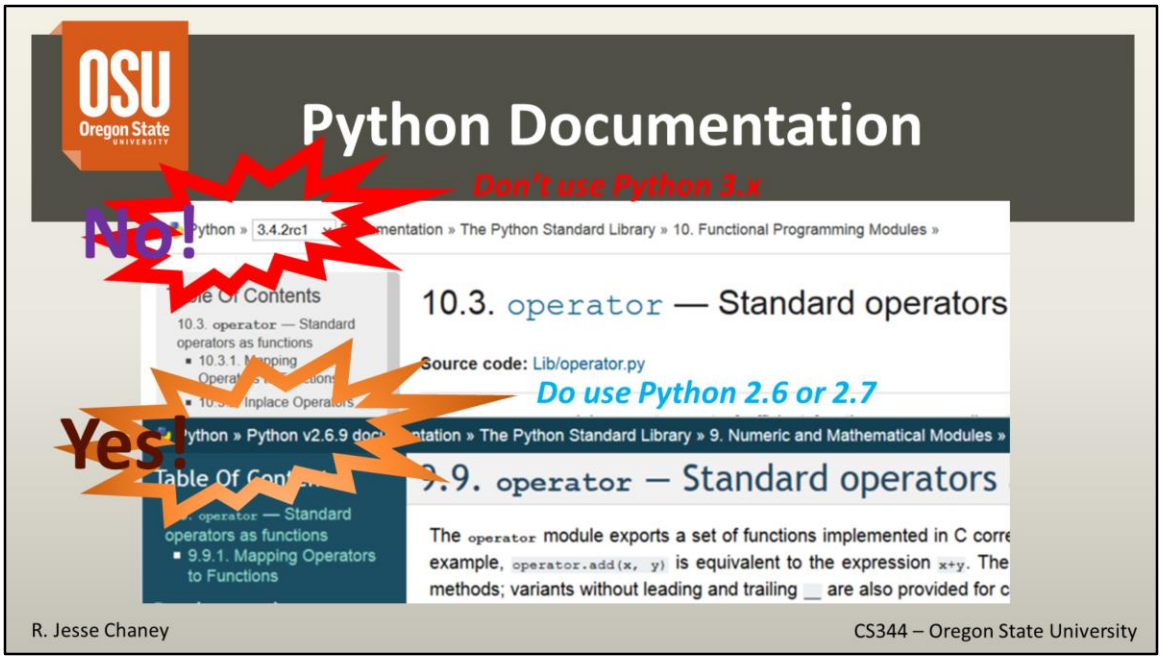
CS344 – Oregon State University

Though Python started out as a scripting language, it use has grown steadily and has become a popular general purpose programming language.

NASA uses Python. Public Broadcasting uses Python. There are probably several of you who use Python in some on-campus activities. You probably know Python better than I do.

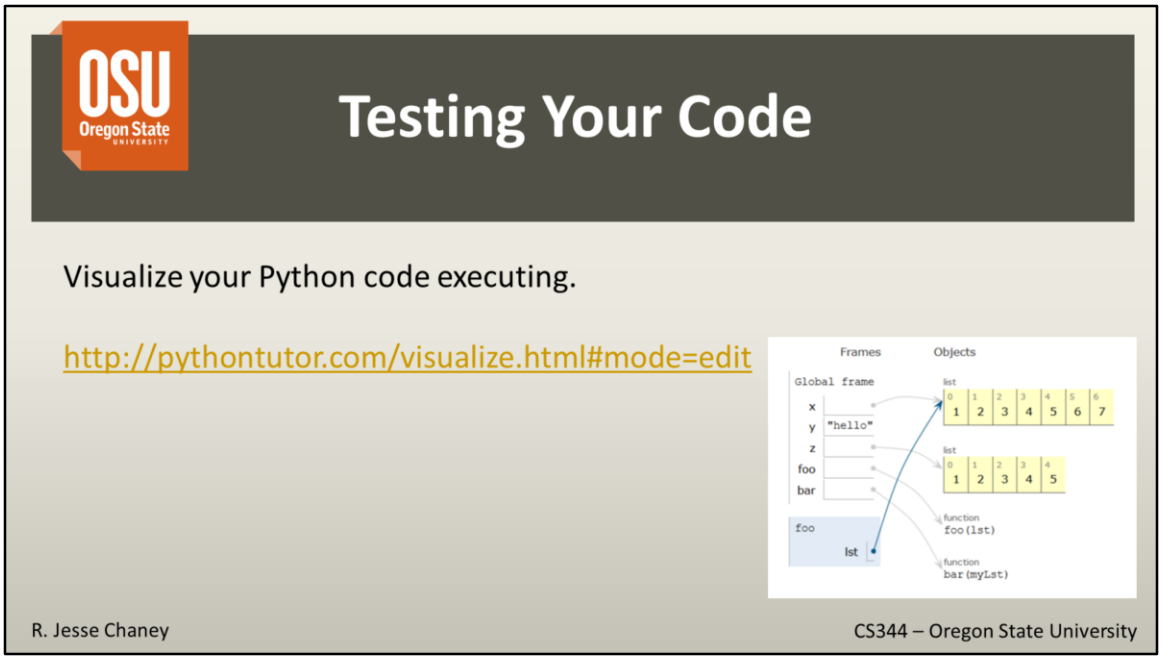
And, let's be serious, that last line there will keep it popular for a while.

The Python web site has a longer list of success stories about Python.
https://www.python.org/about/success



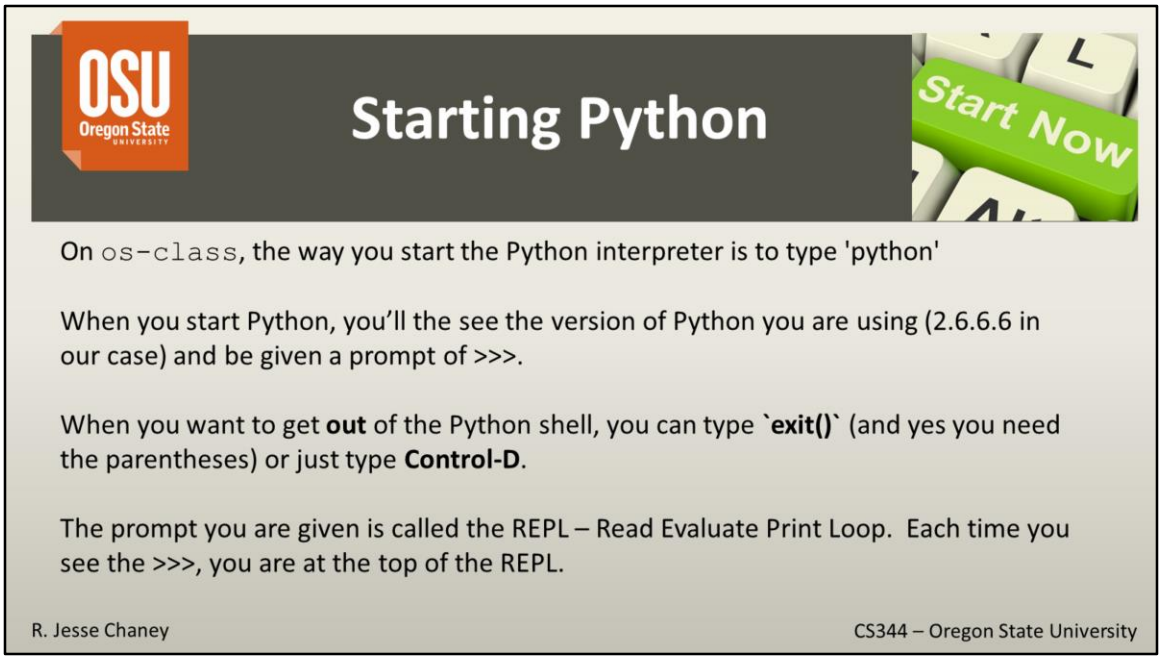
One things you want to be careful about when you lookup information and documentation about Python on the web. Make sure you are looking at documentation for the **right** version of Python.

The official Python web site has documentation for several differing versions of Python. You can find the version as a pull down menu at the upper left of your browser. They've also updated the color scheme to show differently between Python 2 and 3. Don't get all frustrated and wound-up wondering why the examples in the documentation are sewing errors at you only to find out several lumps of hair later that you were looking at docs for Python version 3.4.



This is a really cool web site. In addition to having a tutorial about Python (which I've not read), it has this great tool that allows you to visualize how your Python code is executing.

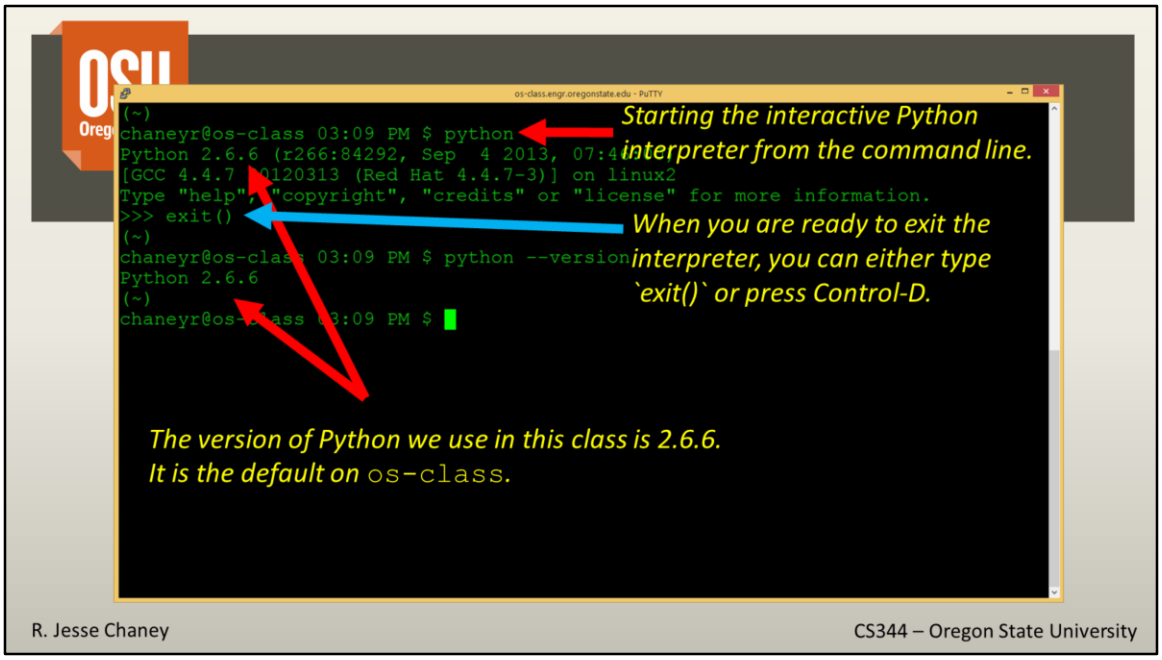
I recommend you try out this web site and see how much it helps you through the Python programming for this class.



If you want to know the version of Python without starting the entire interpreter, you can just type 'python –version`. Make sure you use the double dashes.

Unless you change things around with your PATH environment variable, the default Python version you see will be 2.6.6.6, which is the right one for this class.

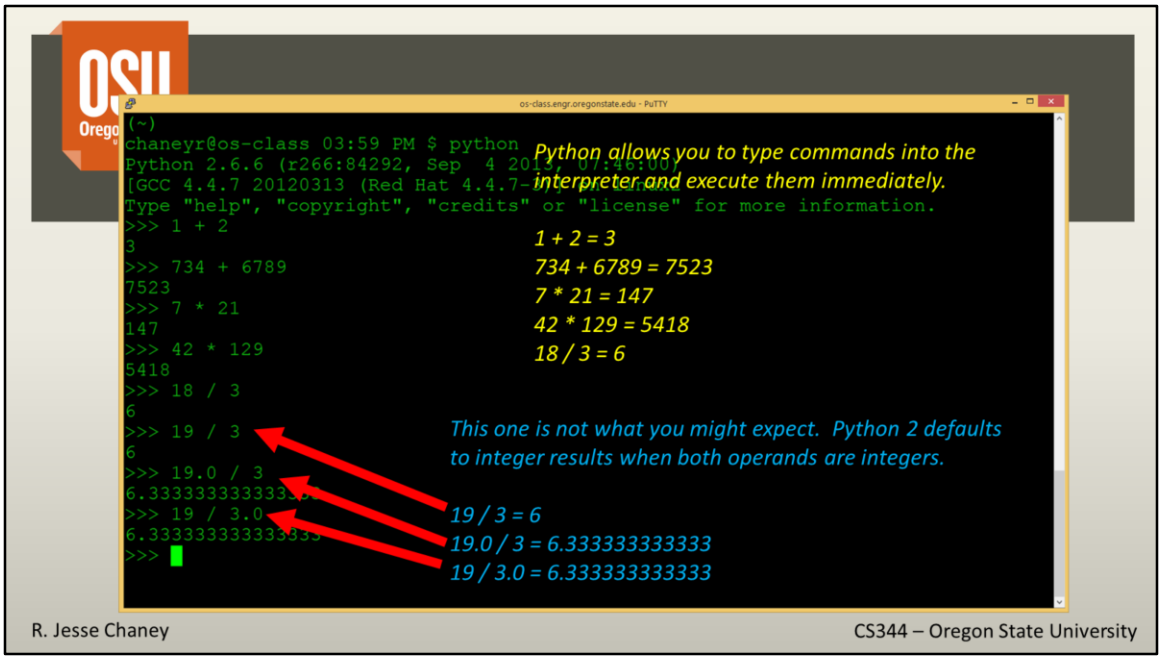
If you are going to use your own Linux system on which to do your development, make sure you install Python 2. Later versions of Python 2 should be fine, but of course you'll test your final code back on `os-class`.



The prompt you are given is called the REPL – Read Evaluate Print Loop. Each time you see the >>>, you are at the base of the REPL.

If you want to know the version of Python without starting the entire interpreter, you can just type 'python –version`. Make sure you use the double dashes.

If you are going to use your own Linux system on which to do your development, make sure you install Python 2. Later versions of Python 2 should be fine, but of course you'll test your final code back on `os-class`.

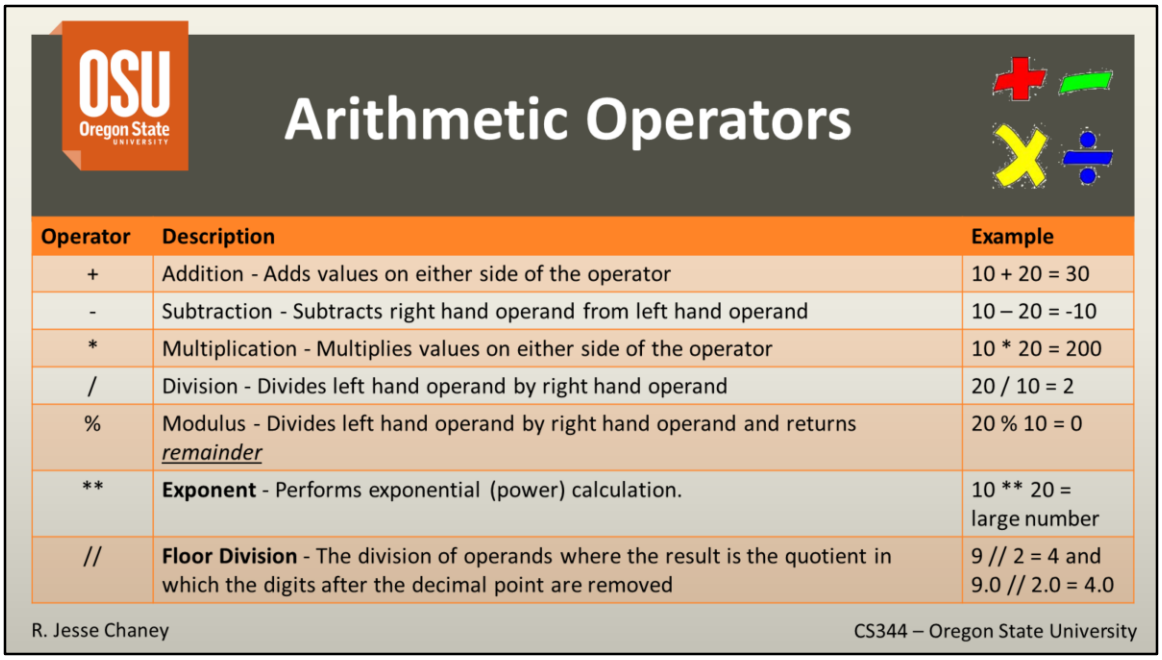


Not surprisingly, you can do simple math at the Python prompt.

1 + 2 = 3
734 + 6789 = 7523
7 * 21 = 147
42 * 129 = 5418
18 / 3 = 6

Al things you'd pretty much expect

19 / 3 = 6 is probably not what you'd expect. This is one the differences between Python 2 and Python 3. By default, with Python 2, when both operands are integers, the result is an integer as well. If you want the floating point result, at least one of the operands needs to be a floating point number.
19.0 / 3 = 6.33333333333 either one or both of the operands can be a floating point value to give a floating point result.
19 / 3.0 = 6.33333333333



Python has the arithmetic operators you'd expect. You might take notice of the exponentiation operator and the floor division operator.

Unlike C, you don't have the ++ increment and the – decrement operators.

Comparison Operators

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(10 == 20) is False
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true	(10 != 20) is True
<>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true. This is like the != operator.	(10 <> 20) is True
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(10 > 20) is False
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(10 < 20) is True
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(10 >= 20) is False
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(10 <= 20) is True

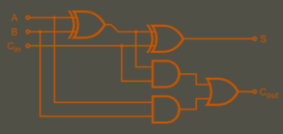
The comparison operators are again what you'd expect. However, notice that you actually have 2 inequality operators, != and <>.

Assignment Operators

Operator	Description	Example
=	Simple assignment operator	c = a + b
+=	Add and assign operator	c += a
-=	Subtract and assign operator	c -= a
*=	Multiply and assign operator	c *= a
/=	Divide AND assign operator	c /= a
%=	Modulus and assign operator	c %= a
**=	Exponent and assign operator	c **= a
//=	Floor Division divides and assigns a value	c //= a

These are operators you've seen many times. The exponent and assign and floor division and assign might be new to you.

Bitwise Operators



Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(a & b)
|	Binary OR Operator copies a bit if it exists in either operand.	(a | b)
^	Binary **XOR Operator** copies the bit if it is set in one operand but not both.	(a ^ b)
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~ b)
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	(a << b)
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	(a >> b)

The only thing you really want to notice here is that the ^ (caret or circumflex) operator is an exclusive or operator, not exponentiation.

Other Operators

Operator	Description	Example
and	Logical and operator. If both the operands are true then then condition becomes true.	(a > b) and (b > c)
or	Logical or operator. If any of the two operands are non zero then then condition becomes true.	(a > b) or (b > c)
not	Logical not operator. Use to reverses the logical state of its operand. If a condition is true then Logical not operator will make false.	not (a > b)

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	a is b
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	a is not b

The Python logical operators are a little be different in that the logical and operator is actually and. The logical or operator is or. Finally, the logical not operator is not.

Identity operators compare the memory locations of two objects.

There are a couple of operators for membership that we'll review a bit later on.

Built-in Constants

False
The false value of the bool type.

True
The true value of the bool type.

None
The only value in the types.NoneType module. *None* is frequently used to represent the absence of a value, as when default arguments are not passed to a function.

Susan Constant

R. Jesse Chaney

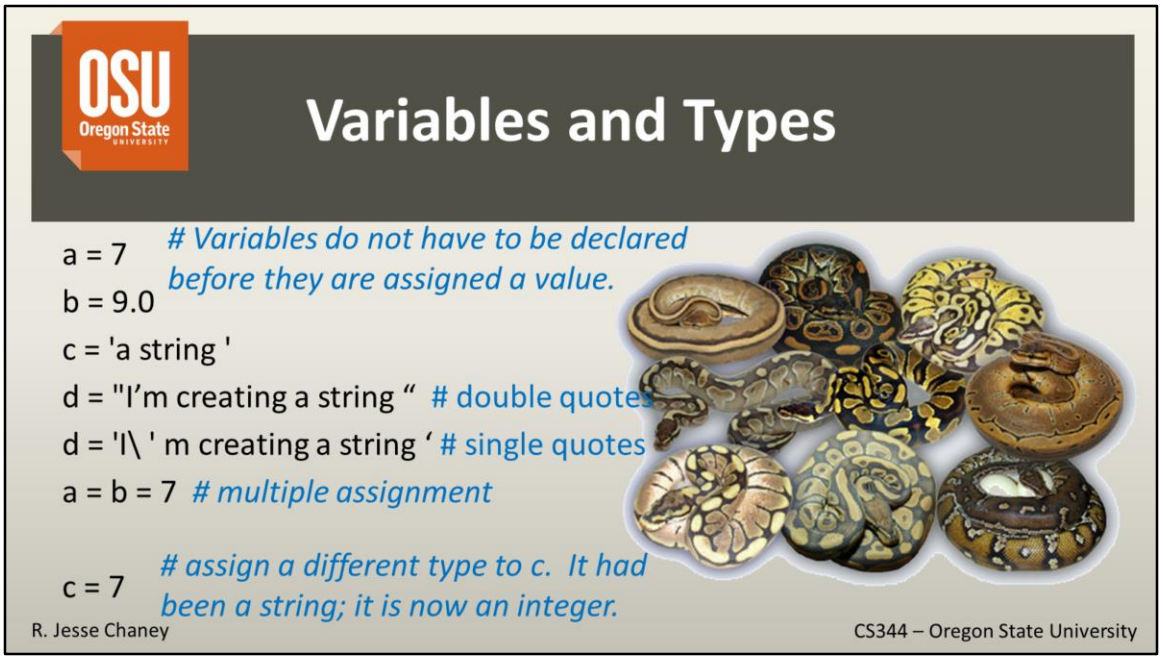
CS344 – Oregon State University

You'll notice that the constant Boolean values True and False begin with a capital letter.

You can also use zero as False. Like in C, everything that is not False is True.

A special value available in Python is None, indicating that the value of the variable is "unassigned". The None in Python is used in same way as null is used in other languages.

BTW, the ship is the *Susan Constant*, the largest of the three ships that landed at Jamestown Colony in 1607.



In Python, it's easy to create variables. You simply assign a value to a name. Unlike Perl and PHP, you don't need to put a dollar sign into EVERY variable name. You don't need to predefine a variable. Just assign a value to it.

Strings can be created using either single or double quotes, so long as the quotes are balanced of the same kind (single or double). In the second example of the strings, you see how you can embed a single quote within a double quoted string. You can also escape the apostrophe if you want to use single quotes to create your string. You can find more about how to assign strings in Python manual 3.1.2.

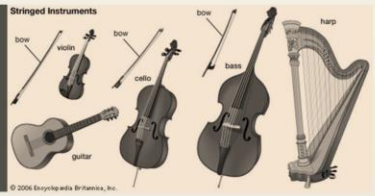
Notice in the last line that I assign a string to a variable which had just be created to contain an integer. This is how you get dynamic typing. You can assign any type to any variable at any time.

In addition to integers, floating point, and strings, Python has full support for complex numbers.

In addition to these, we are going to cover lists and maps types. Python has additional types: tuples, classes, Unicode strings, sequences, sets, and more. You'll

need to go through the docs on your own for those.

Strings




```
s = 'this string \n\
spans lines '
s = ' ' '
this string
spans lines
' ' '
s = "this " + "is " + "concatenation "
s = "this " "is " "also " "a " "concatenation "
```

 # A string that spans lines, using the continuation character.

 # A string that spans lines, within the triple quote characters. Continuation char not needed.

 # This only works with string literals, omit the + sign

R. Jesse Chaney

CS344 – Oregon State University

Assignments to strings can span multiple lines, using the continuation character as the last character on the line (a \)

You can use triple single or double quotes to define large blocks of text.

The + operator can be use to concatenate strings. You can also simply place string literals next to each other to concatenate them.

There is not a character data type. A character is a string of length 1.

Strings

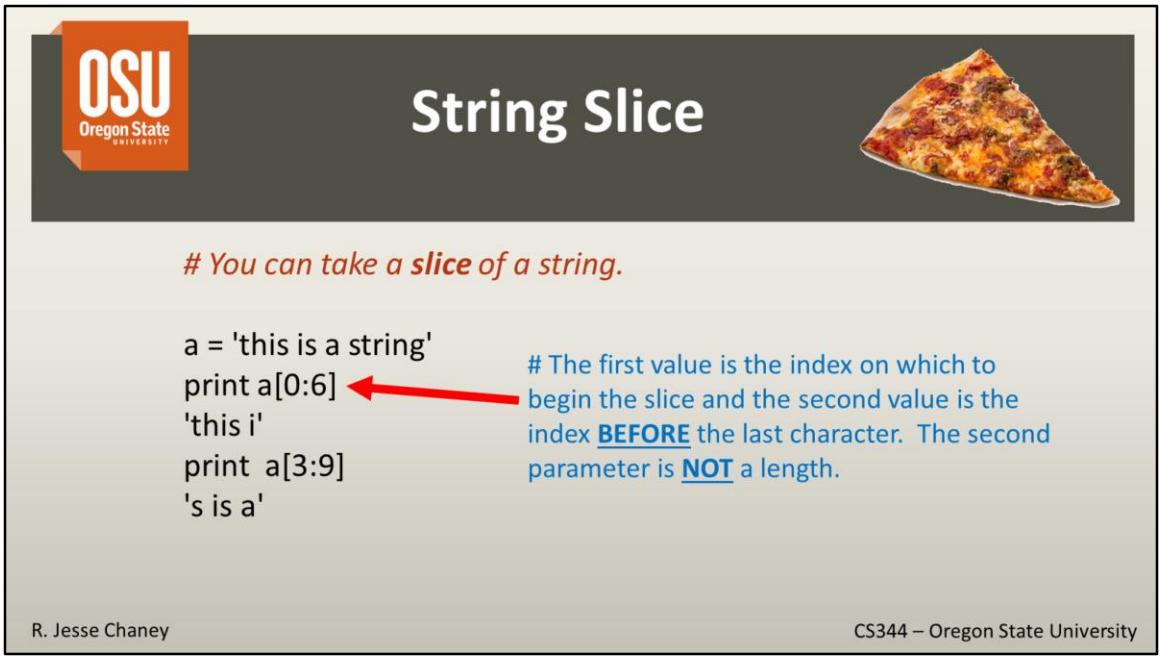
*# Strings can be subscripted, just like in C. Just as in C, the strings are indexed **starting with 0**.*

```
a = 'this is a string'
print a[0]
't'
print  a[5]
'i'
print len(a)
16
```

 # The initial character in a string is index 0.

 # The 5th index is what we would call the 6th character.

 # The length of the string.



You can access a group of characters in a string by specifying a range, a number a colon and another number. This is called a slice. Slices are an important concept in Python. In the notation for a slice, the first value is the index on which to begin the slice and the second value is the index **BEFORE** the last character.

Slices are a big deal in Python.

More String Slices

a = 'this is a string'

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
character	t	h	i	s		i	s		a		s	t	r	i	n	g

Slice 3 to 9

Slice notation of *str[n:m]*
The slice a[3:9] is 's is a'
The slice begins with the nth character (zero based) and ends with the character ***before*** the mth character. It is **NOT** a length parameter.

R. Jesse Chaney CS344 – Oregon State University

If this think of the locations as being before or between characters in the string, it is a little easier to see how a[3:9] is “s is a” and not “s is a “

Still More Slices

 # You can take slices is other ways.

```
a = 'this is a string'
print a[:6]
'this i'
print a[6:]
's is a string'
```

 # If you omit the first argument, it defaults to 0.

 # If you omit the second argument, it defaults to the rest of the string.

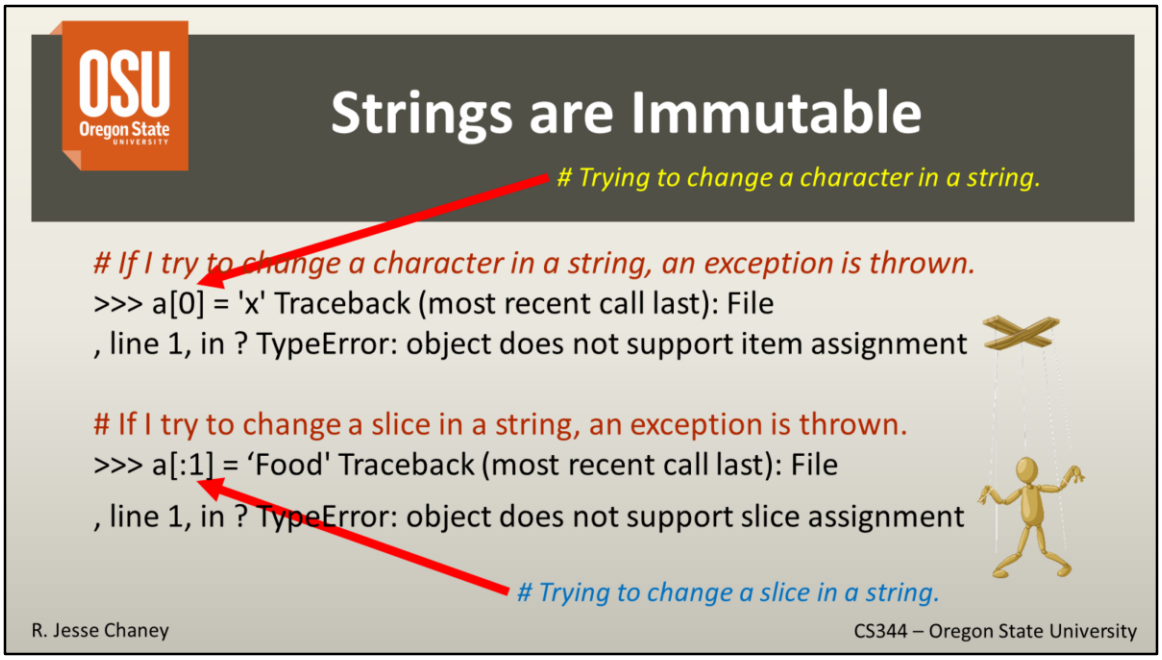
Slices from the End

```
# You can take slices starting from the end of the list.
a = 'this is a string'
print a[-2]
' n'
print a[-2:]
'ng'
print a[:-2]
'this is a stri'
```

*# Work from the **end of the string**, the second to last character.*

*# The second to last character to the **end**.*

*# Beginning to **before** the second to last character.*



Unlike a C string, Python strings are *immutable; you can't change he contents of a string*. Assigning a new value to an indexed position in the string results in an error. You can, of course, assign a new string value to a varable.

Trying to assign a new value to a slice also returns an error.

You can assign a new value to a string, but you cannot assign to a substring, as you can in C.

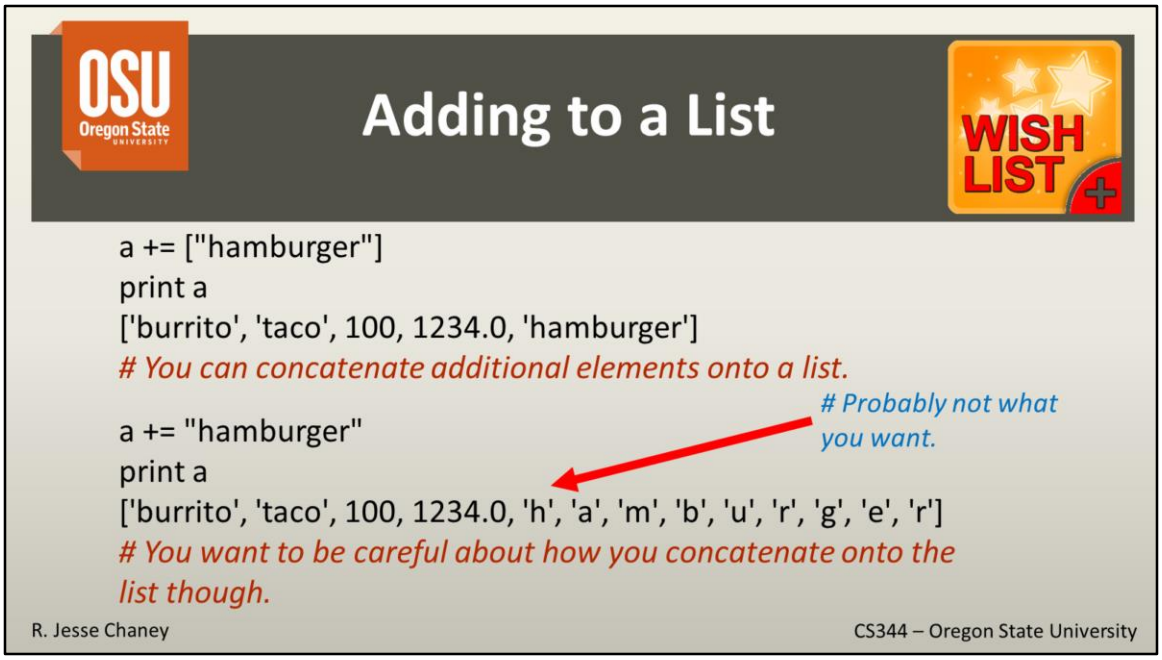
Lists

```
# A list can contain items of different types.
a = ["burrito", 'taco', 100, 1234.0]
# The elements of a list do not all have to be of the same type.

print a[0]
'burrito'
# Just as with strings, the initial element of a list is index 0.

print a[1:]
['taco', 100, 1234.0]
# You can takes slices of lists just as you can with strings.
```

R. Jesse Chaney

CS344 – Oregon State University

You use square bracket notation to create lists in Python.



Notice the difference between adding the list [“hamburger”] and adding the string “hamburger”

Lists within Lists


```
a = ["burrito", 'taco', 100, 1234.0]
a += [[4, 5, 6, 7]]
print a
['burrito', 'taco', 100, 1234.0, [4, 5, 6, 7]]
# Adding a list within a list.

print a[4]
[4, 5, 6, 7]
# This time, printing a single member of the list is a list.
```

 # Adding a list nested within list.

 # A list nested within another list.

Lists are not Immutable

```
# Starts off with yellow mustard.
a = ['yellow mustard', 'catsup', 'mayonnaise', 'bbq']
a[0] = 'deli mustard'
print a
# Substitute with deli mustard.
['deli mustard', 'catsup', 'mayonnaise', 'bbq']
print a[0]
deli mustard
# You can make substitutions in a list.
```

More List Creation

```
a = ["burrito", 'taco', 100, 4 * 5, 3.0 / 6]
print a
['burrito', 'taco', 100, 20, 0.5]
# You can create a list using expressions.

import math
a = ["burrito", 'taco', 100, math.pi]
print a
['burrito', 'taco', 100, 3.1415926535897931]
# Another kind of expression in a list creation.
```

 # Expression for assignment.

 # A value from another module.

R. Jesse Chaney

CS344 – Oregon State University

You can use Python expression when you create a list.

We'll talk more about the import function in a little bit.

Range Function

```
a = range(10)
print a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

 # A list of 10 numbers, 0 through 9.

```
a = range(3, 15)
print a
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

*# As with strings and lists, the range ends just **before** the end-value. Start with 3 and continue to (15 -1).*

```
a = range(3, 60, 17)
print a
[3, 20, 37, 54]
```

*# The range function also an **optional step** argument. Use the function as range(begin, end, step)*

Stranger Ranger

```
a = range(-10, 1)
print a
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0]
```

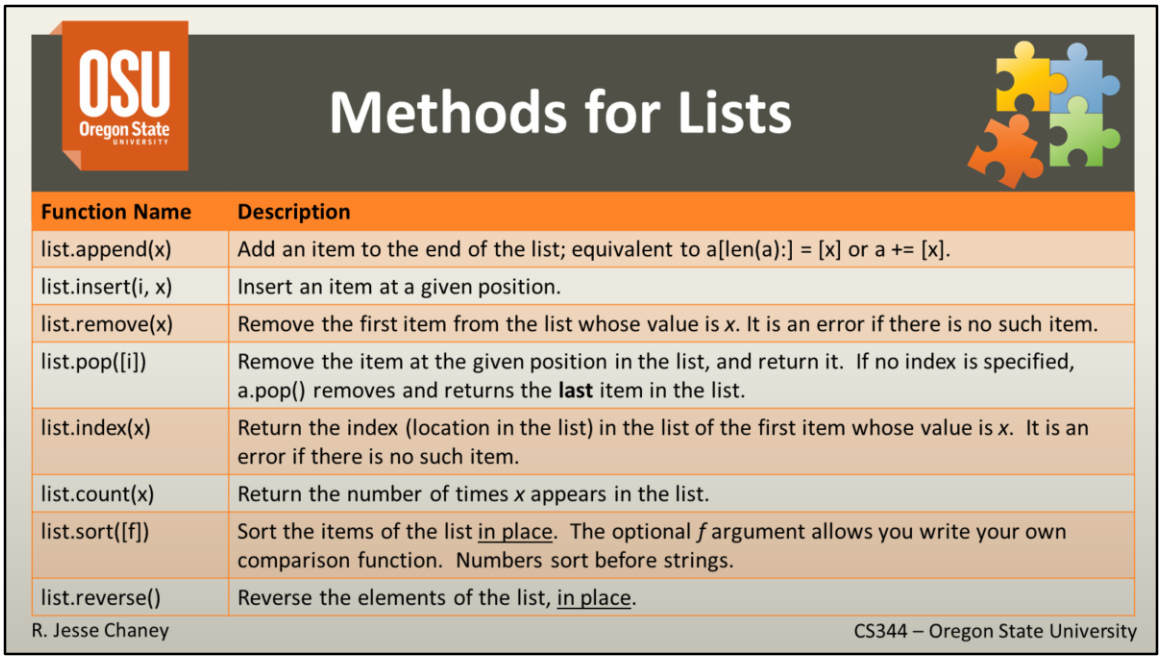
 # Start at -10 and continue to just before 1.

```
a = range(-10, -20, -1)
print a
[-10, -11, -12, -13, -14, -15, -16, -17, -18, -19]
```

 # Start at -10 and continue to just before -20, decrementing by -1.

```
a = range(-10)
print a
[]
```

 # Probably not what you are looking for, an empty list.



There are many built-in functions you can apply to lists. Here are a few of them.

You can treat lists as stacks, queues, etc.

If it has not come up yet, lists can be nested as well.

Dictionary

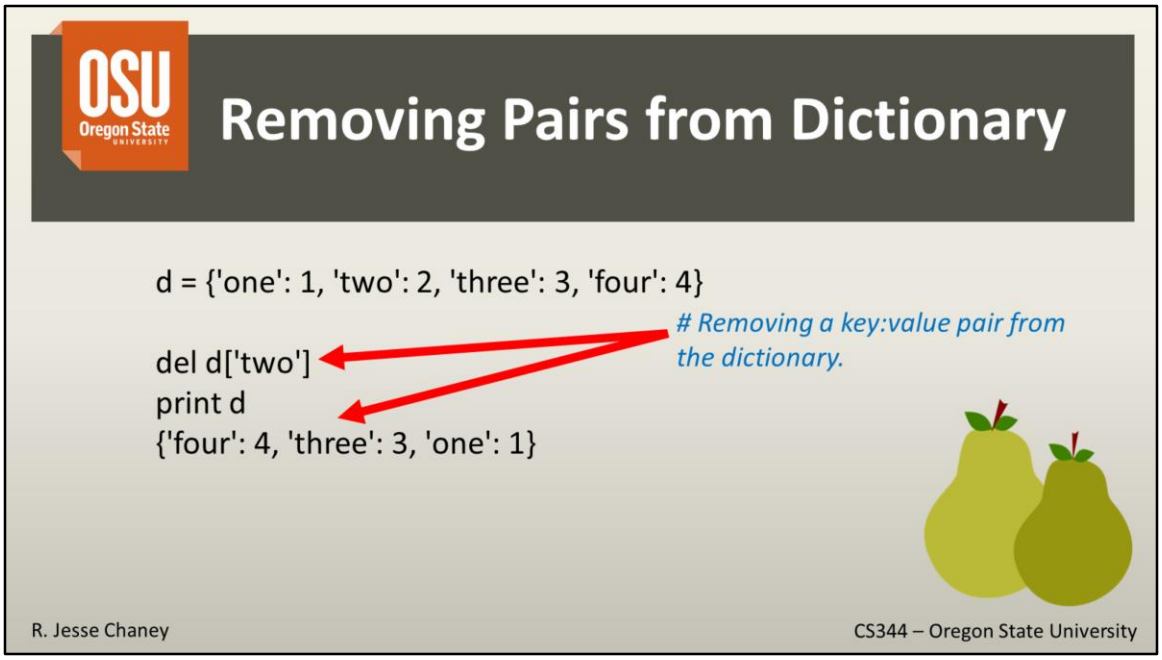
```
# An unordered set of key: value pairs,
# keys must be unique.
d = {'one': 1, 'two': 2, 'three': 3, 'four': 4}
print d['three']
3

# Adding a new entry to the dictionary.
d['five'] = 5
# Note how the order
# is different from how
# it was created.
print d
{'four': 4, 'three': 3, 'five': 5, 'two': 2, 'one': 1}
```

R. Jesse Chaney

CS344 – Oregon State University

Another important type in Python is a map or dictionary. A dictionary is an unordered set of *key: value* pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}. Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary.

You can add pairs to a dictionary.



You can delete pairs in a dictionary.

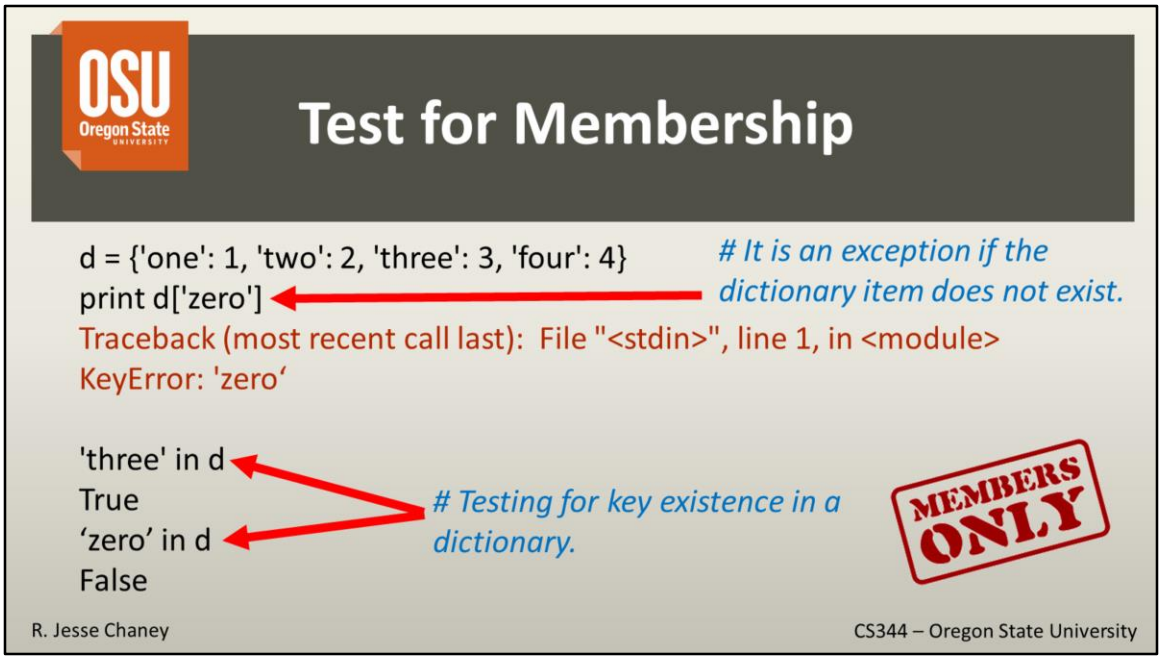
Keys and Values

```
d = {'one': 1, 'two': 2, 'three': 3, 'four': 4}

print d.keys()                    # Returns list of the keys in the dictionary.
['four', 'three', 'two', 'one']

print d.values()                  # Returns list of the just the values
[4, 3, 2, 1]                      # in the dictionary.
```

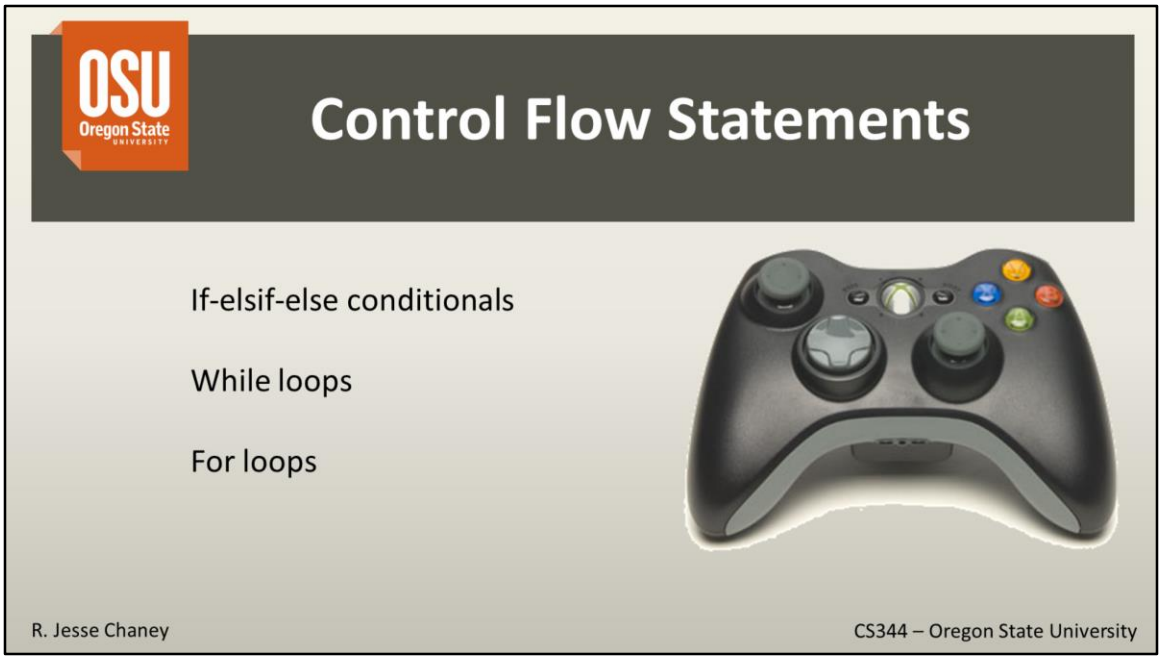
R. Jesse Chaney

CS344 – Oregon State University

You can list all the keys of a dictionary and you can list all the values in a dictionary.



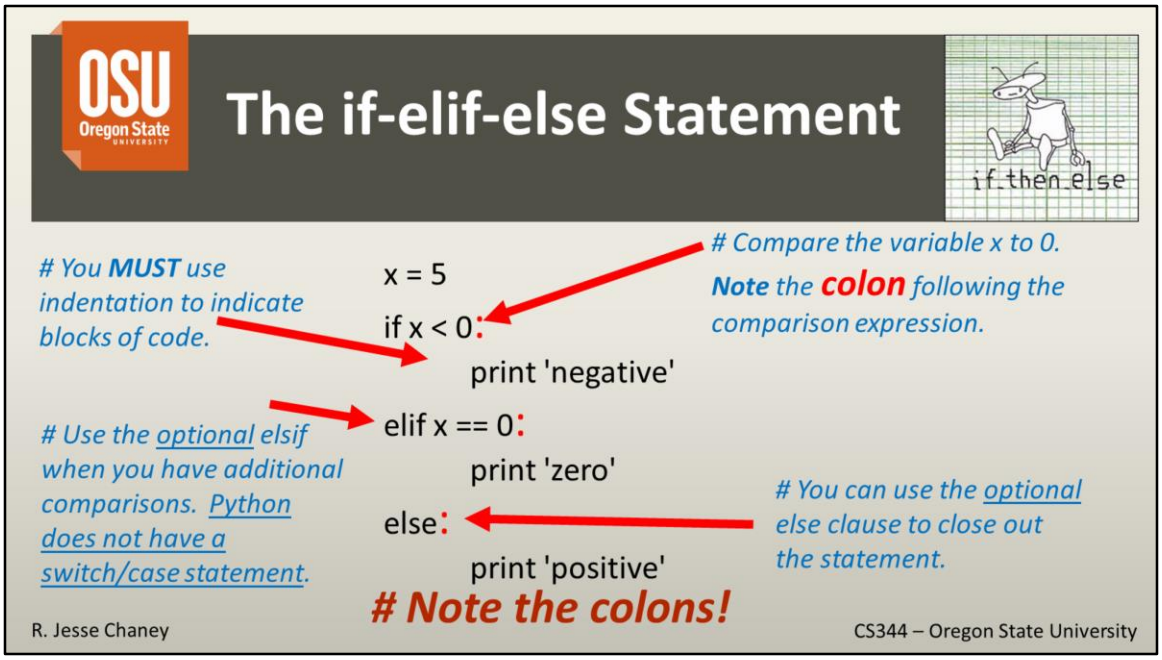
If you try to access pair that does not exist, it is an error. However, there is an operator, in, that can be used to test for membership. I alluded to this operator early in the slide collection.



Python has the normal set of control flow structures.

If-elsif-else, while loops, for loops (which are more like foreach loops).

There are break and continue statements that you can combine with them.



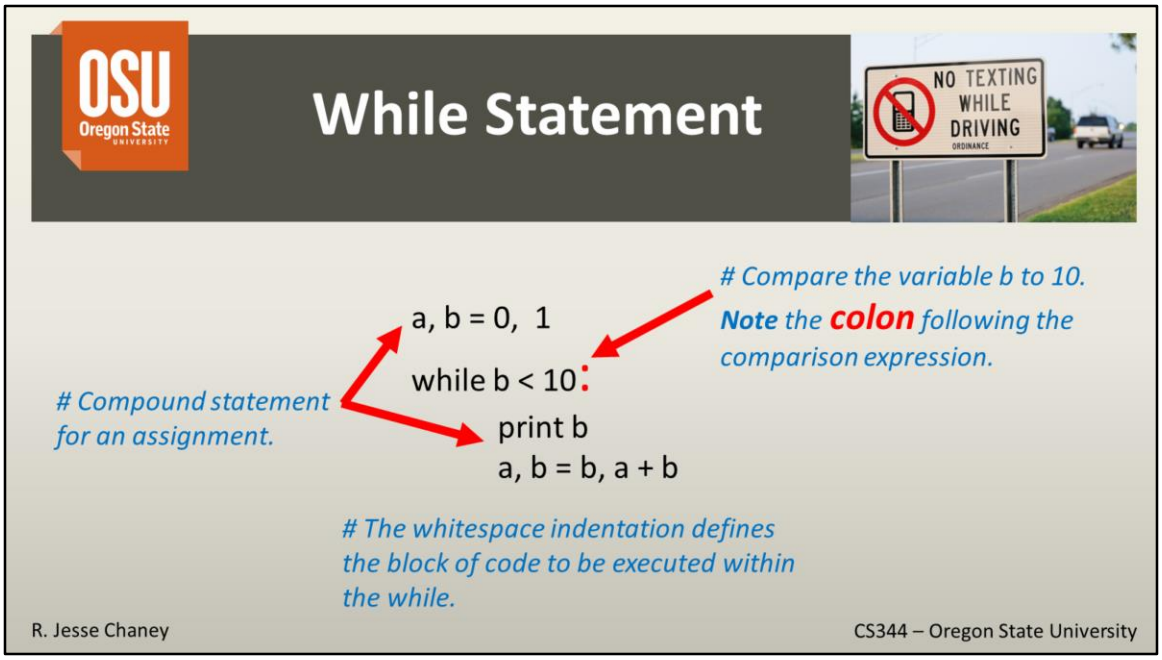
This is a busy slide.

We start an **if** statement comparing x to 0. Note that the conditional does not have to be enclosed within parentheses.
Also, notice the **colon** that follows the conditional. It is very important and you'll probably forget it a few times (like I did when I started using Python). If you omit the colon, you'll get an error statement, "IndentationError: expected an indented block"

As you can see, if I want to have more than one conditional within the statement, I use the reserved word elif. The elif is helpful at avoiding some excessive indentation.

You should also be aware that Python does not have a switch-case statement. You'll use the if-elif-else statement where you'd normally think to use a switch-case statement.

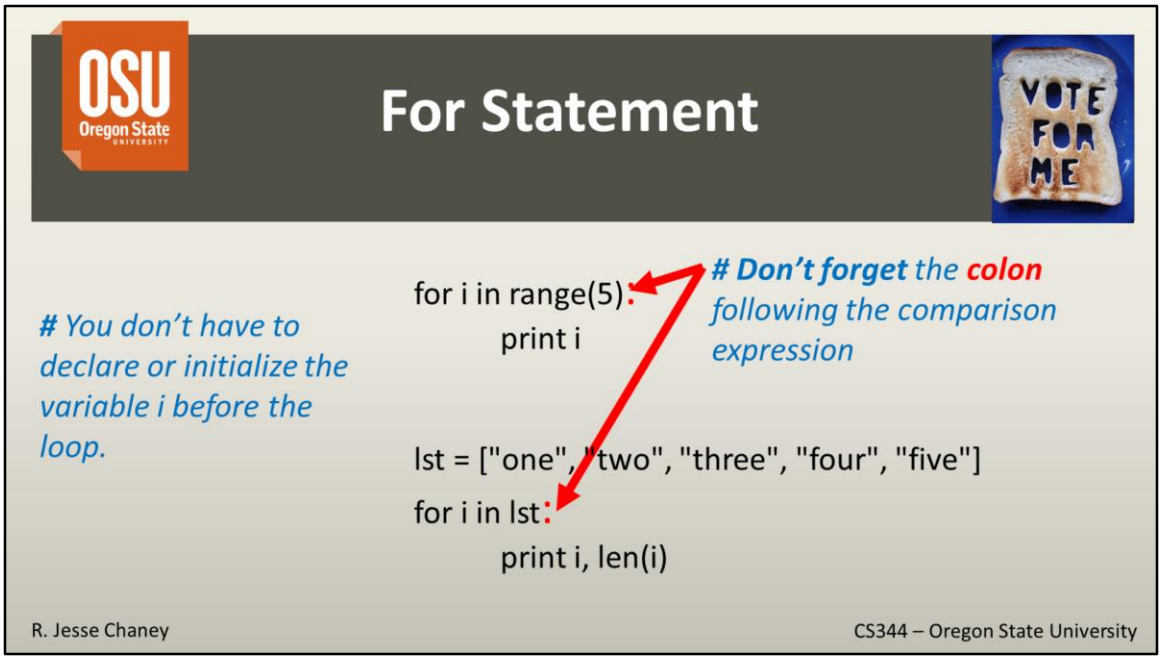
You close of the block of code within the clauses by simply returning to the higher level indentation level. If you've never been good about indenting your code before Python, you'll get better because of it.



The initial line of Python code shows a compound statement for an assignment. This is probably something that is new. It is very convenient.

As with the if-statement, the conditional of the while statement does not need to be enclosed with parentheses. The colon does need to follow it though.

In this example, you see a little bit better how the whitespace defines the code within the while block.



When you need to iterate over a range of numbers or through the items in a list, the for loop is built for you. It is really more like a foreach loop.

Notice that the variable i gets assigned each value in turn from the list of items provided to the for loop. That is why it is often called a foreach loop.

The break Statement

```
for letter in 'Hamburger':
   if letter == 'b':
      break
   print 'Current Letter :', letter

var = 10
while var > 0:
   print 'Current variable value :', var
   var = var -1
   if var == 5:
      break
```

 # The for loop will break when the 'b' from "Hamburger" is found.

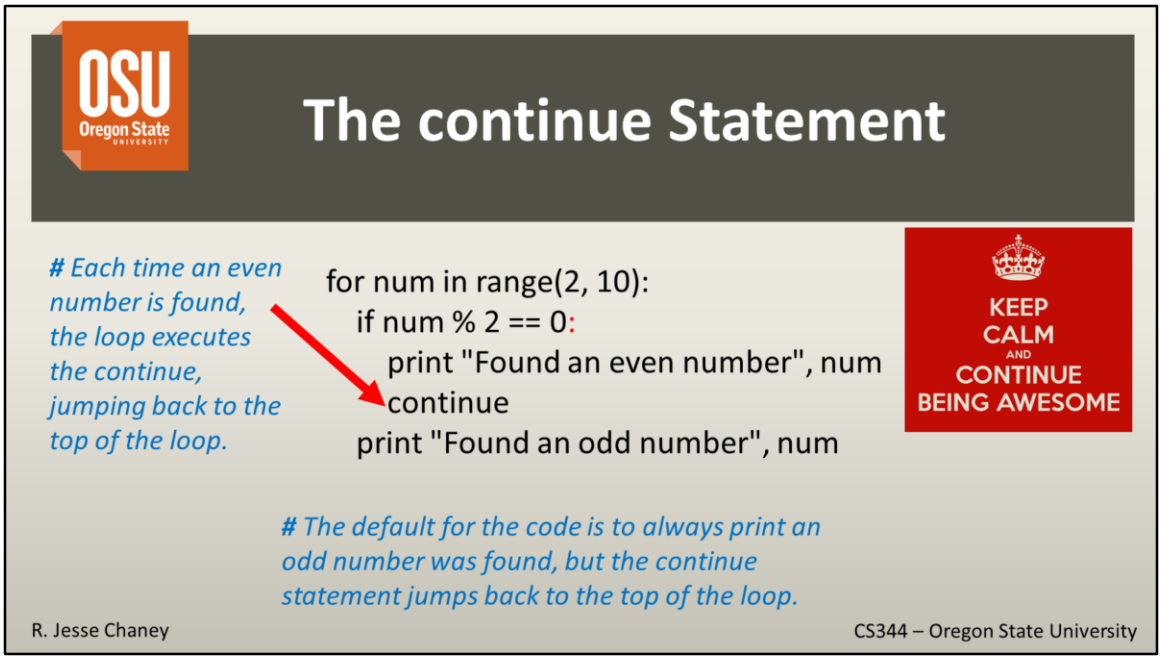
 # The while loop will break when the value of the var variable is equal to 5.

R. Jesse Chaney

CS344 – Oregon State University

The *break* statement in Python is like the break statement in C, it breaks out of the **innermost** enclosing *for* or *while* loop.

These are great pieces of code to paste into the visual execution web page.



The *continue* statement in Python also behave like it's C counterpart; it continues with the next iteration of the for/while loop.

This is a great piece of code to paste into the visual execution web page.

The pass Statement

```
for letter in 'Burrito':
  if letter == 'r':
    pass
    print 'This is pass block'
print 'Current Letter :', letter
```

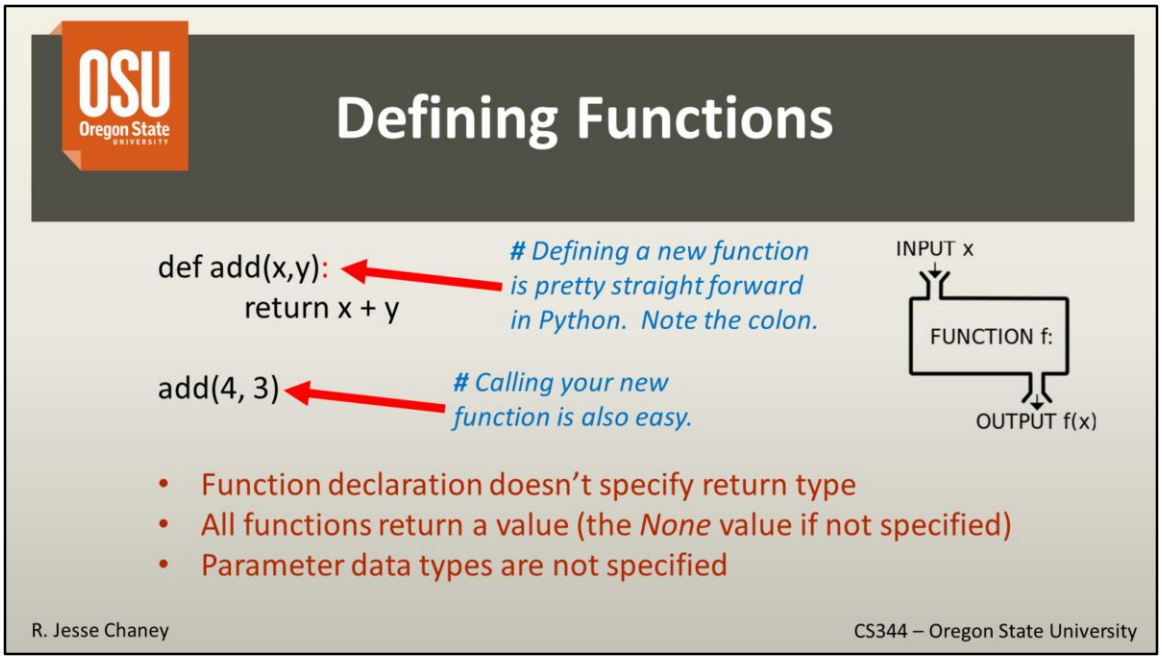
 # The pass statement is executed each time the letter 'r' is encountered.

R. Jesse Chaney

CS344 – Oregon State University

The *pass* statement does nothing. It can be used when a statement is required syntactically but the program requires no action.

This is a great piece of code to paste into the visual execution web page.

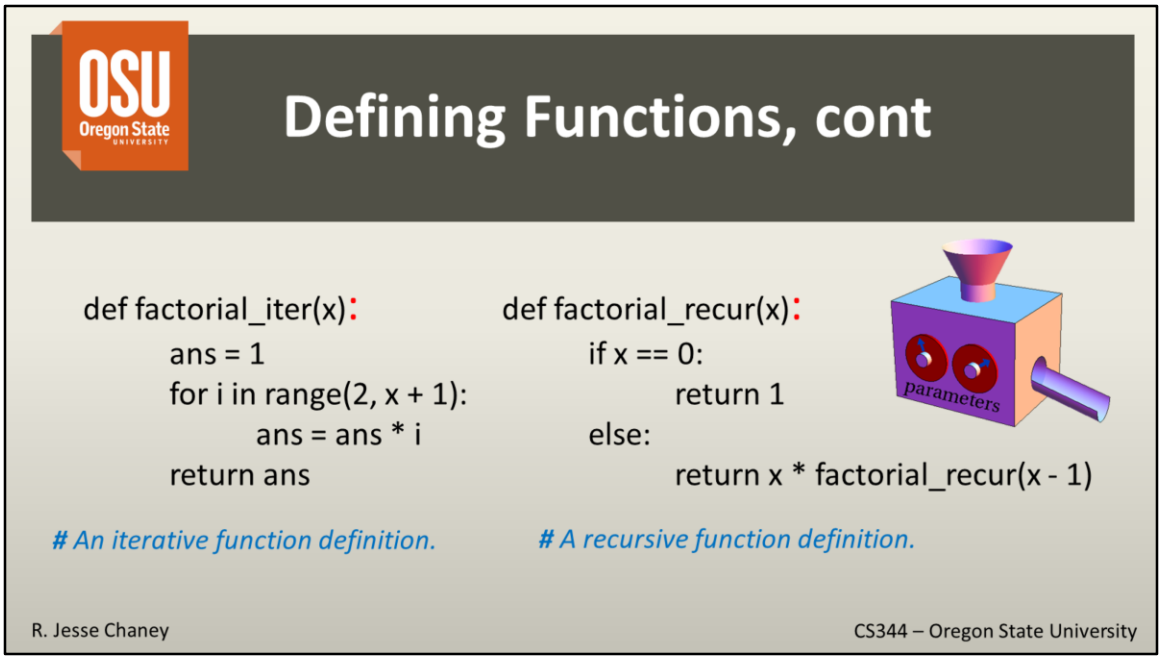


After seeing how you create while and for loops, defining functions should look pretty natural.

You have to make sure you use the colon at the end of the line where you use the def keyword. And you use whitespace indentation for the entire function body.

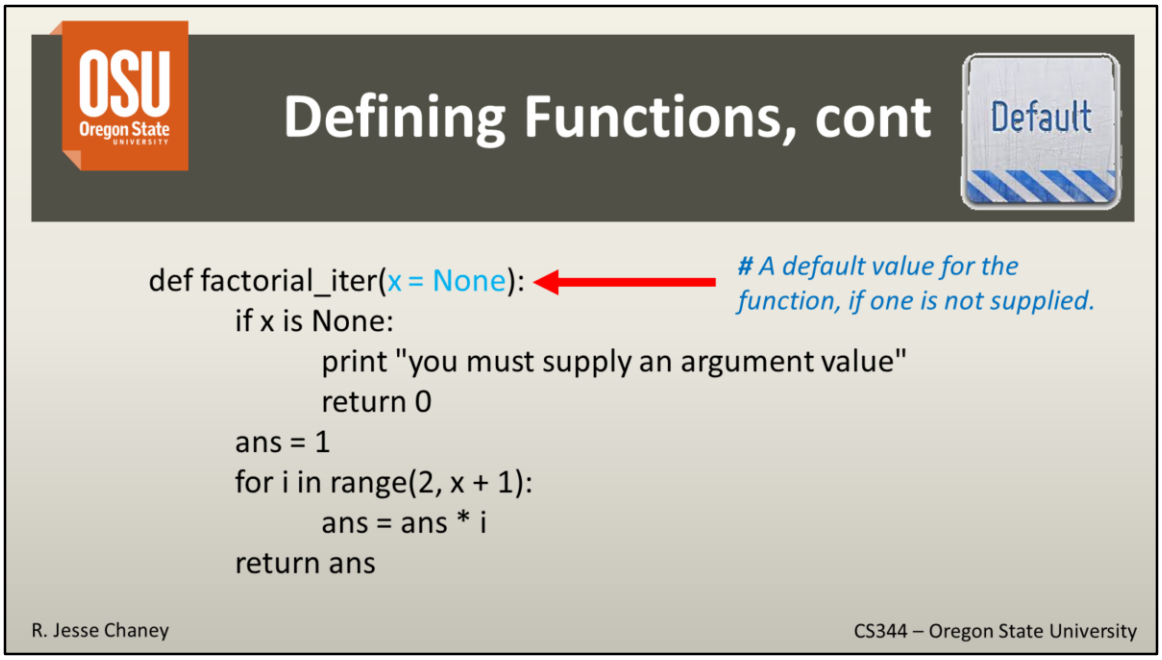
I think one of the goals of Python is to get us to use better indentation styles and make pretty code.

```
def add(x,y):
                return x + y

add(3, 4)
add(17, 56)
```



Just as a means to compare, here are 2 examples showing how to define an iterative implementation of the factorial function and a recursive implementation of factorial.

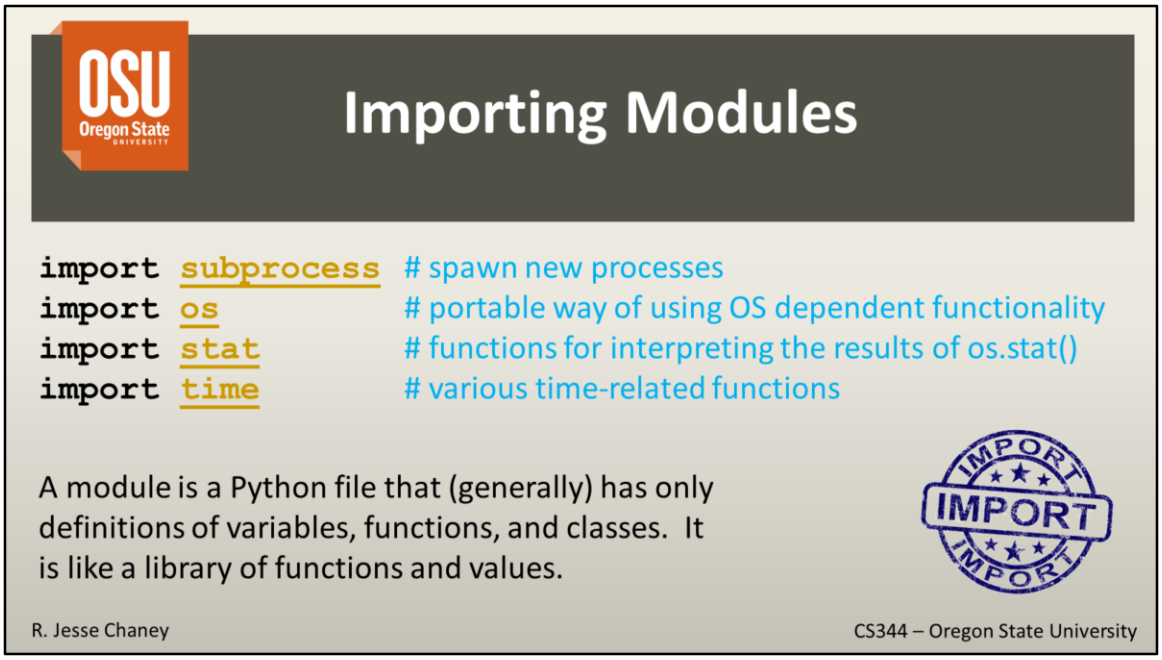
Obviously, there are a lot of extra checks that could be put in this code.



When you define a new function, it is easy for you to provide default values to the function arguments.

There are additional ways to name arguments when calling a function. You can consult the docs about those.

```
def factorial_iter(x = None):
        if x == None:
                print "you must supply an argument"
                return 0
        ans = 1
        for i in range(2, x + 1):
                ans = ans * i
        return ans

factorial_iter()
```



A module is a python file that (generally) has only definitions of variables, functions, and classes

There are many Python modules that you can use. In order to access the functions in those other modules, you need to `import` the module into your file. A very common one to import is the math module.

By default, when you import a module, you have to prefix any use of variables or functions within that module with the module name. However, you can use the `from <module_name> import <name>` syntax if you want to avoid the prefix. Just make sure that names in the module don't conflict with other names you may be using.

If you want to know the list of names defined within a module, you can use the built-in function `dir(<module_name>)`. Of course, you are probably better off going to the documentation to see what each name stands for.

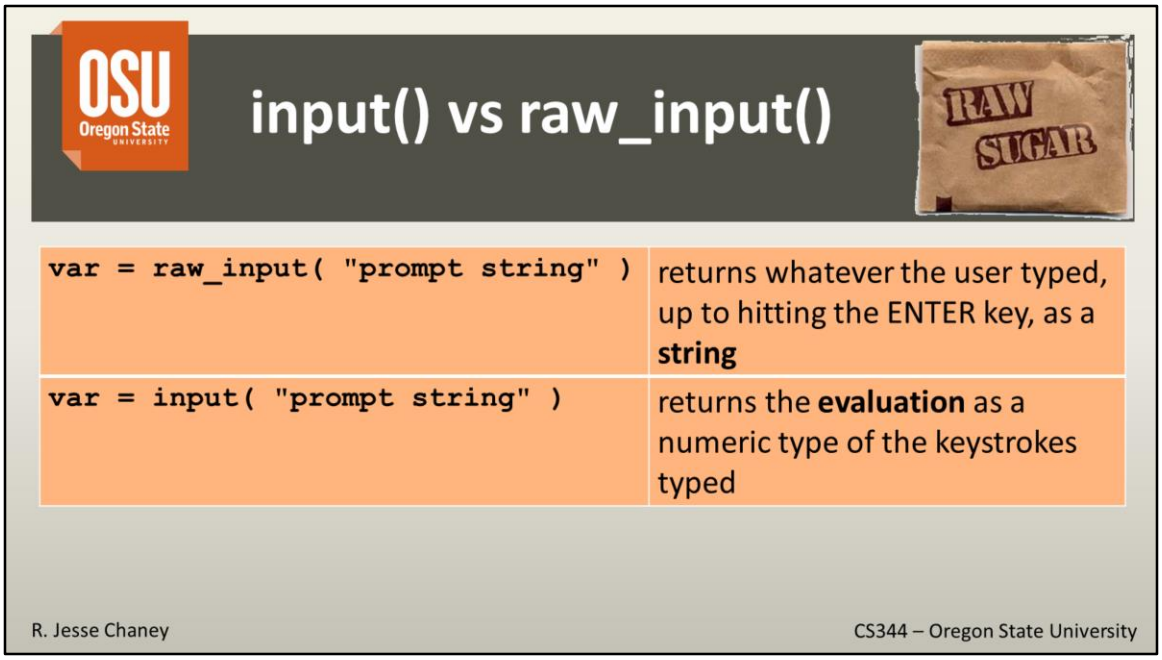
You can go further an divide modules into packages. I'll leave it to you to consult the docs for that.

As you develop larger Python applications, you can break up your code into multiple modules.

```
import math
print math.pi
```

```
pi
```

```
from math import pi
pi
```



If you are reading input from the keyboard, you need to be aware of the difference between the input() function and the raw_input() function. The value returned from input() must be a valid Python expression and it returns the evaluated result to you.

str = input("Enter your input: ")

str = raw_input("Enter your input: ")

The print statement

```
print
print value1, value2, value3
print "this is easy to do", value1
print "This is a %s formatted string" % str
print "this is formatting a number %2d" % value1
print "This can be used to print the number %d" % value1
print "do not put a new line at the end",
```

R. Jesse Chaney

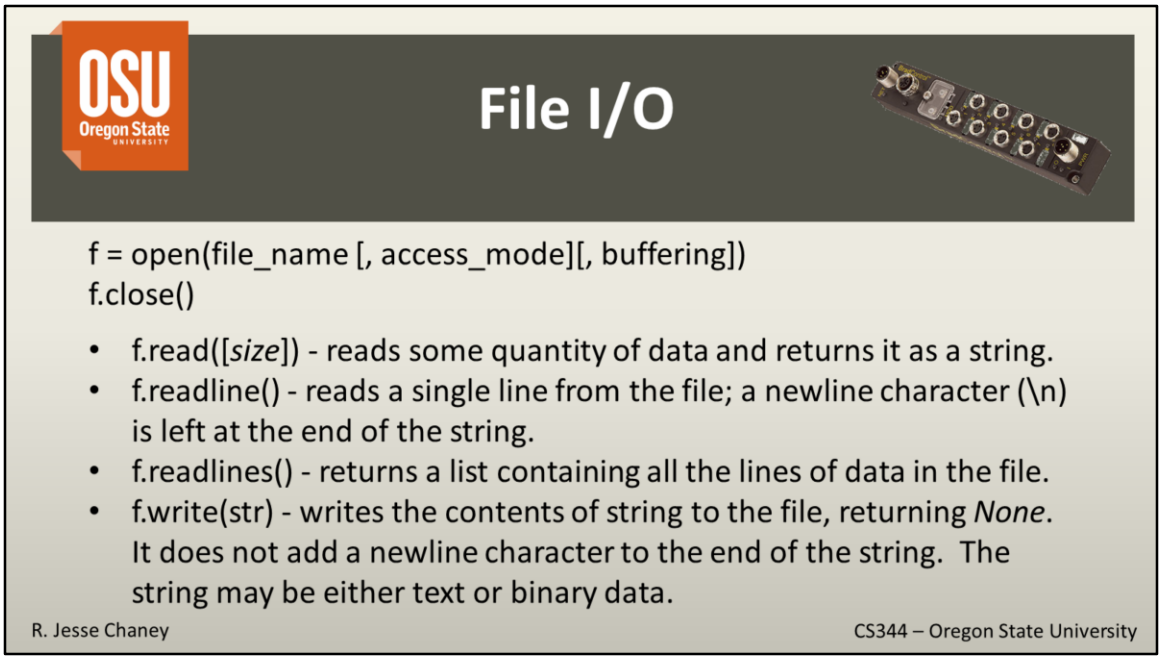
CS344 – Oregon State University

The simplest way to produce output is using the *print* statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output. A comma between arguments with the print function places a single space between them. The line is terminated with a newline, unless a comma is the last thing on the line.

If you want to format the string a bit more, you can use the str.rjust(#) and str.ljust(#) methods. You specify the preferred width of the output field. If the argument happens to be wider than the given width, the output is not truncated.

If you want more elegant printed output, you can make use of the format function, in section 7 of the Python documentation.

Python also supports a C style % use. This is an older style and the format function should be preferred over it.



If you are performing file i/o, you have to first call the open() function to open the file. There are several different mode for opening a file: read-only, write-only, read-write, binary read, binary write, append, append binary, …

Obviously, if you have to open the file, you'll need to also close it.

The file object has a couple methods for reading data from a file: read(), readline(), readlines().

Though there are a couple methods to read from a file, there is only 1 to write to a file. The write method writes a string to the file. The string can be binary data or character data.

You are going to get to know the Unix/C version of open() and close() much better than you ever thought comfortable.

Exception Handling

```
while True:
     try:
          x = int(raw_input("Please enter a number: "))
          break
     except ValueError:
          print "That was not valid number.  Try again..."
```

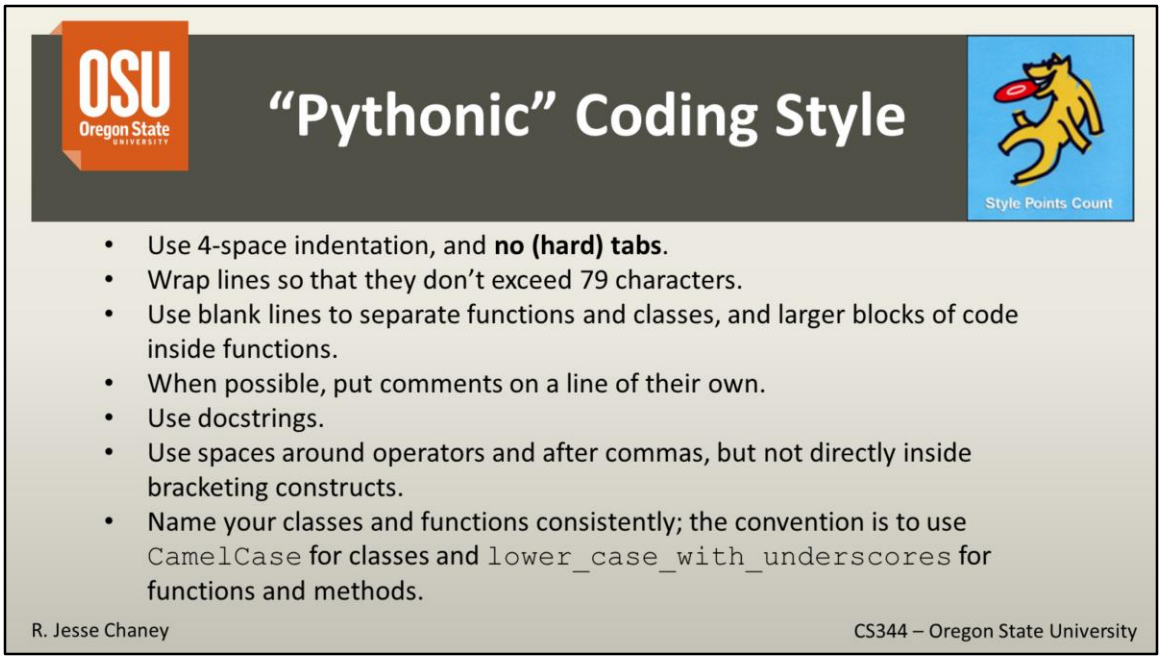
R. Jesse Chaney

CS344 – Oregon State University

Remember one of the things I miss about C? I miss exception handling (try/catch block). Well, Python has them.

Just as in C++ and Java (and other languages), Python has exception handling.

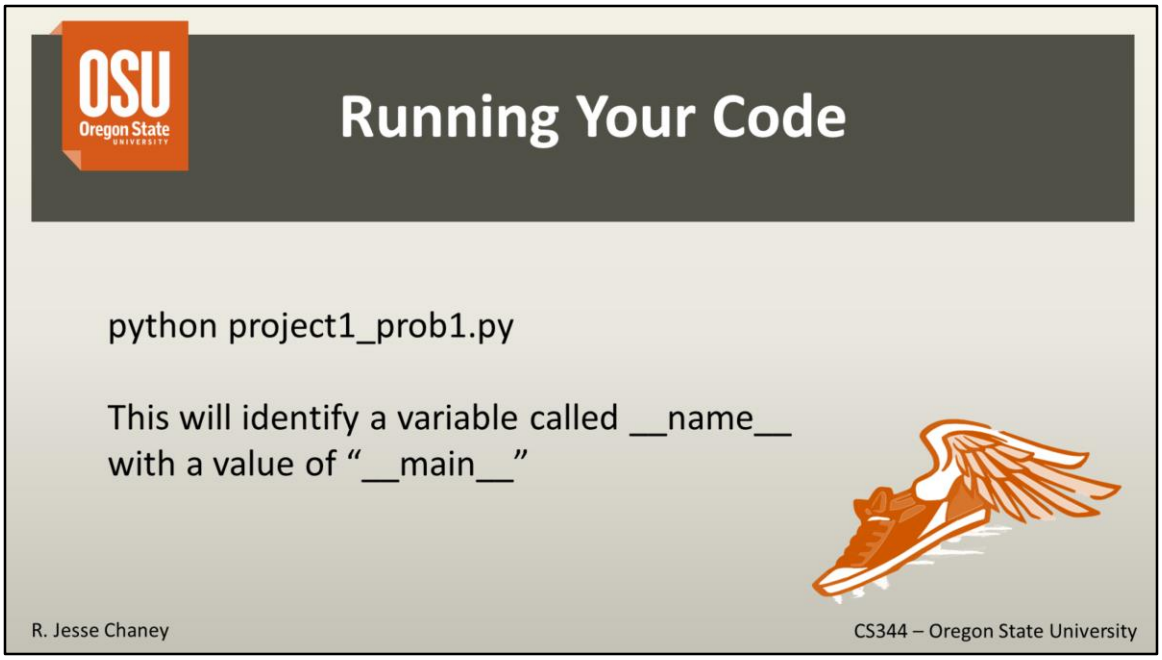
```
while True:
        try:
                x = int(raw_input("Please enter a number: "))
                break
        except ValueError:
                print "Oops!  That was no valid number.  Try again..."
```



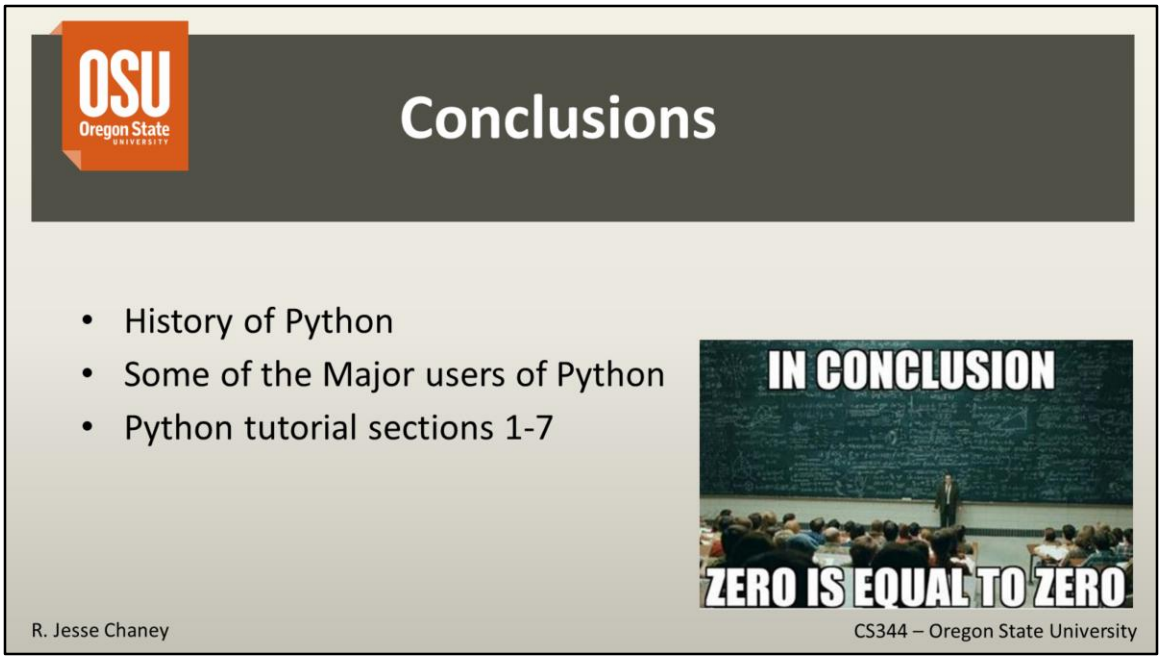
I'm not a big stickler on coding style. I've used a lot of coding styles in my 25+ years of writing code. Some of them I liked, some I did not. Python has developed a coding style called PEP 8. **One of the ideas is that code is read much more often than it is written, so making it readable is worthwhile.**

One of the other important phrases of the doc guide is "**A Foolish Consistency is the Hobgoblin of Little Minds".** Don't be consistent if it makes things worse.

- 4 spaces are a good compromise between small indentation (allows greater nesting depth) and large indentation (easier to read). Tabs introduce confusion, and are best left out.
- This helps users with small displays and makes it possible to have several code files side-by-side on larger displays.
- Don't use fancy encodings if your code is meant to be used in international environments. Plain ASCII works best in any case.



Running your Python script is pretty easy once you get the code into a .py file.



There is a lot more about Python. This is just an introduction.

At this point, you are probably hungry for more, or just hungry.