



Introduction to Python

Python

- It is a scripting language like Perl or PHP.
- Sort of a compromise between shell scripts and C/C++/Java/...
- It is an interpreted language, not compiled (normally)
- It is Dynamically typed
- Python is Object Oriented



R. Jesse Chaney

CS344 – Oregon State University

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 a 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).

R. Jesse Chaney



CS344 – Oregon State University

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 lose 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, its 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>

No!

OSU
Oregon State University

Python Documentation

Don't use Python 3.x

python » 3.4.2rc1 » Documentation » The Python Standard Library » 10. Functional Programming Modules »

Table Of Contents

- 10.3. `operator` — Standard operators as functions
 - 10.3.1. Mapping Operators
 - 10.3.2. Inplace Operators

Source code: [Lib/operator.py](#)

Do use Python 2.6 or 2.7

python » Python v2.6.9 documentation » The Python Standard Library » 9. Numeric and Mathematical Modules »

Table Of Contents

- 9.9. `operator` — Standard operators as functions
 - 9.9.1. Mapping Operators to Functions

9.9. `operator` — Standard operators

The `operator` module exports a set of functions implemented in C corresponding to the built-in operations in Python. For example, `operator.add(x, y)` is equivalent to the expression `x+y`. The methods; variants without leading and trailing `_` are also provided for c

R. Jesse Chaney

CS344 – Oregon State University

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.

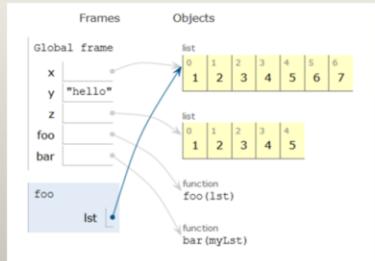
The official Python web site has documentation for several differing versions of Python. You can find the version as a pull down menu at the upper left of your browser. They've also updated the color scheme to show differently between Python 2 and 3. Don't get all frustrated and wound-up wondering why 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.



Testing Your Code

Visualize your Python code executing.

<http://pythontutor.com/visualize.html#mode=edit>



R. Jesse Chaney

CS344 – Oregon State University

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.



Starting Python



On `os-class`, the way you start the Python interpreter is to type '`python`'

When you start Python, you'll see the version of Python you are using (2.6.6.6 in our case) and be given a prompt of `>>>`.

When you want to get **out** of the Python shell, you can type ``exit()`` (and yes you need the parentheses) or just type **Control-D**.

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

R. Jesse Chaney

CS344 – Oregon State University

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 screenshot shows a terminal window titled "os-class.engr.oregonstate.edu - PuTTY". The window displays the following text:

```
chaneyr@os-class 03:09 PM $ python
Python 2.6.6 (r266:84292, Sep 4 2013, 07:41:02)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.

>>> exit()
(~)
chaneyr@os-class 03:09 PM $ python --version
Python 2.6.6
(~)
chaneyr@os-class 03:09 PM $
```

Annotations over the terminal text:

- A red arrow points from the text "Starting the interactive Python interpreter from the command line." to the line "chaneyr@os-class 03:09 PM \$ python".
- A blue arrow points from the text "When you are ready to exit the interpreter, you can either type 'exit()' or press Control-D." to the line ">>> exit()".
- A red arrow points from the text "The version of Python we use in this class is 2.6.6. It is the default on os-class." to the line "chaneyr@os-class 03:09 PM \$ python --version".

The terminal window has a dark background and light-colored text. The annotations are color-coded to highlight specific parts of the text.

R. Jesse Chaney

CS344 – Oregon State University

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.

```
OSU
Oregon
(chaneyr@os-class 03:59 PM) $ python
Python 2.6.6 (r266:84292, Sep  4 2011, 07:46:00)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 + 2
1 + 2 = 3
>>> 734 + 6789
734 + 6789 = 7523
>>> 7 * 21
7 * 21 = 147
>>> 42 * 129
42 * 129 = 5418
>>> 18 / 3
18 / 3 = 6
>>> 19 / 3
6
>>> 19.0 / 3
6.333333333333333
>>> 19 / 3.0
19 / 3 = 6
19.0 / 3 = 6.333333333333333
19 / 3.0 = 6.333333333333333
```

This one is not what you might expect. Python 2 defaults to integer results when both operands are integers.

R. Jesse Chaney

CS344 – Oregon State University

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
```

All things you'd pretty much expect

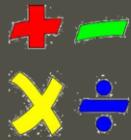
$19 / 3 = 6$ is probably not what you'd expect. This is one the differences between Python 2 and Python 3. By default, with Python 2, when both operands are integers, the result is an integer as well. If you want the floating point result, at least one of the operands needs to be a floating point number.

$19.0 / 3 = 6.333333333333333$ either one or both of the operands can be a floating point value to give a floating point result.

$19 / 3.0 = 6.333333333333333$



Arithmetic Operators



Operator	Description	Example
+	Addition - Adds values on either side of the operator	$10 + 20 = 30$
-	Subtraction - Subtracts right hand operand from left hand operand	$10 - 20 = -10$
*	Multiplication - Multiplies values on either side of the operator	$10 * 20 = 200$
/	Division - Divides left hand operand by right hand operand	$20 / 10 = 2$
%	Modulus - Divides left hand operand by right hand operand and returns <i>remainder</i>	$20 \% 10 = 0$
**	Exponent - Performs exponential (power) calculation.	$10 ** 20 =$ large number
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed	$9 // 2 = 4$ and $9.0 // 2.0 = 4.0$

R. Jesse Chaney

CS344 – Oregon State University

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

R. Jesse Chaney

CS344 – Oregon State University

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

R. Jesse Chaney

CS344 – Oregon State University

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 operand's value is moved left by the number of bits specified by the right operand.	(a << b)
>>	Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.	(a >> b)

R. Jesse Chaney

CS344 – Oregon State University

The only thing you really want to notice here is that the \wedge (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 the condition becomes true.	$(a > b) \text{ and } (b > c)$
or	Logical or operator. If any of the two operands are non zero then the condition becomes true.	$(a > b) \text{ or } (b > c)$
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.	$\text{not } (a > b)$

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

R. Jesse Chaney

CS344 – Oregon State University

The Python logical operators are a little 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.



Variables and Types

```
a = 7 # Variables do not have to be declared  
      before they are assigned a value.
```

```
b = 9.0
```

```
c = 'a string'
```

```
d = "I'm creating a string" # double quotes
```

```
d = 'I\ ' m creating a string' # single quotes
```

```
a = b = 7 # multiple assignment
```

```
c = 7 # assign a different type to c. It had  
      been a string; it is now an integer.
```

R. Jesse Chaney



CS344 – Oregon State University

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 been created to contain an integer. This is how you get dynamic typing. You can assign any type to any variable at any time.

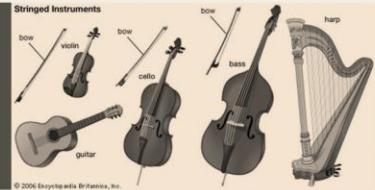
In addition to integers, floating point, and strings, Python has full support for complex numbers.

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 '  
# A string that spans lines, using  
# the continuation character.  
  
s = '''  
this string  
spans lines  
'''  
  
s = "this " + "is " + "concatenation "  
s = "this " "is " "also " "a " "concatenation "  
# 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 used 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] # The initial character in a string is index 0.  
't'  
print a[5] # The 5th index is what we would call the  
          # 6th character.  
'i'  
print len(a) # The length of the string.  
16
```



String Slice



You can take a *slice* of a string.

```
a = 'this is a string'  
print a[0:6] ←  
'this i'  
print a[3:9]  
's is a'
```

The first value is the index on which to begin the slice and the second value is the index **BEFORE** the last character. The second parameter is **NOT** a length.

R. Jesse Chaney

CS344 – Oregon State University

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 notation of $str[n:m]$

The slice $a[3:9]$ is 's is a'

The slice begins with the n^{th} character (zero based) and ends with the character before the m^{th} 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 in other ways.

```
a = 'this is a string'  
print a[:6] # If you omit the first argument, it defaults to 0.  
'this i'  
print a[6:] # If you omit the second argument, it defaults  
# to the rest of the string.  
's is a string'
```



R. Jesse Chaney

CS344 – Oregon State University



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.



Strings are Immutable

Trying to change a character in a string.

If I try to change a character in a string, an exception is thrown.

```
>>> a[0] = 'x' Traceback (most recent call last): File  
, line 1, in ? TypeError: object does not support item assignment
```



If I try to change a slice in a string, an exception is thrown.

```
>>> a[:1] = 'Food' Traceback (most recent call last): File  
, line 1, in ? TypeError: object does not support slice assignment
```

Trying to change a slice in a string.



R. Jesse Chaney

CS344 – Oregon State University

Unlike a C string, Python strings are *immutable*; you can't change the 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 variable.

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 take 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.



Adding to a List



```
a += ["hamburger"]
print a
['burrito', 'taco', 100, 1234.0, 'hamburger']
# You can concatenate additional elements onto a list.
a += "hamburger"
print a
['burrito', 'taco', 100, 1234.0, 'h', 'a', 'm', 'b', 'u', 'r', 'g', 'e', 'r']
# Probably not what you want.
# You want to be careful about how you concatenate onto the list though.
```

R. Jesse Chaney

CS344 – Oregon State University

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]] ← # Adding a list nested within list.
print a
['burrito', 'taco', 100, 1234.0, [4, 5, 6, 7]] ← # Adding a list within a list.
# This time, printing a single member of the list is a list.
print a[4]
[4, 5, 6, 7]
```

R. Jesse Chaney

CS344 – Oregon State University



Lists are not Immutable

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

Starts off with yellow mustard.

Substitute with deli mustard.





More List Creation



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

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

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

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

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

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

R. Jesse Chaney

CS344 – Oregon State University



Stranger Ranger



```
a = range(-10, 1) # Start at -10 and continue to just before 1.  
print a  
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0]  
  
a = range(-10, -20, -1) # Start at -10 and continue to just before -20, decrementing by -1.  
print a  
[-10, -11, -12, -13, -14, -15, -16, -17, -18, -19]  
  
a = range(-10) # Probably not what you are looking for,  
print a  
[] # an empty list.
```

R. Jesse Chaney

CS344 – Oregon State University



Methods for Lists



Function Name	Description
list.append(x)	Add an item to the end of the list; equivalent to <code>a[len(a):] = [x]</code> or <code>a += [x]</code> .
list.insert(i, x)	Insert an item at a given position.
list.remove(x)	Remove the first item from the list whose value is <code>x</code> . It is an error if there is no such item.
list.pop([i])	Remove the item at the given position in the list, and return it. If no index is specified, <code>a.pop()</code> removes and returns the last item in the list.
list.index(x)	Return the index (location in the list) in the list of the first item whose value is <code>x</code> . It is an error if there is no such item.
list.count(x)	Return the number of times <code>x</code> appears in the list.
list.sort([f])	Sort the items of the list <u>in place</u> . The optional <code>f</code> argument allows you write your own comparison function. Numbers sort before strings.
list.reverse()	Reverse the elements of the list, <u>in place</u> .

R. Jesse Chaney

CS344 – Oregon State University

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  
  
d['five'] = 5  
print d  
{'four': 4, 'three': 3, 'five': 5, 'two': 2, 'one': 1}
```

Adding a new entry to the dictionary.
Note how the order
is different from how
it was created.



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.

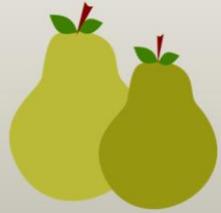


Removing Pairs from Dictionary

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

```
del d['two']
print d
{'four': 4, 'three': 3, 'one': 1}
```

*# Removing a key:value pair from
the dictionary.*



R. Jesse Chaney

CS344 – Oregon State University

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  
in the dictionary.  
[4, 3, 2, 1]
```



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.



Test for Membership

```
d = {'one': 1, 'two': 2, 'three': 3, 'four': 4}      # It is an exception if the  
print d['zero']                                     dictionary item does not exist.
```

```
Traceback (most recent call last): File "<stdin>", line 1, in <module>
```

```
    KeyError: 'zero'
```

```
'three' in d
```

```
True
```

```
'zero' in d
```

```
False
```

Testing for key existence in a
dictionary.



R. Jesse Chaney

CS344 – Oregon State University

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.



Control Flow Statements

If-elsif-else conditionals

While loops

For loops



R. Jesse Chaney

CS344 – Oregon State University

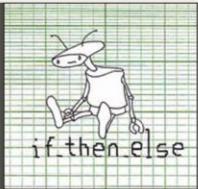
Python has the normal set of control flow structures.

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.



The if-elif-else Statement



You **MUST** use indentation to indicate blocks of code.

Use the optional `elif` when you have additional comparisons. Python does not have a switch/case statement.

R. Jesse Chaney

```
x = 5
if x < 0:
    print 'negative'
elif x == 0:
    print 'zero'
else:
    print 'positive'
```

Compare the variable `x` to 0.

Note the **colon** following the comparison expression.

You can use the optional `else` clause to close out the statement.

Note the colons!

CS344 – Oregon State University

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 off 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.



While Statement



```
a, b = 0, 1  
while b < 10:  
    print b  
    a, b = b, a + b
```

Compound statement for an assignment.

Compare the variable b to 10.
Note the **colon** following the comparison expression.

The whitespace indentation defines the block of code to be executed within the while.

R. Jesse Chaney

CS344 – Oregon State University

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.



For Statement



You don't have to declare or initialize the variable i before the loop.

```
for i in range(5):  
    print i  
  
lst = ["one", "two", "three", "four", "five"]  
for i in lst:  
    print i, len(i)
```

Don't forget the colon following the comparison expression

R. Jesse Chaney

CS344 – Oregon State University

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
```

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



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

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

Each time an even number is found, the loop executes the continue, jumping back to the top of the loop.

```
for num in range(2, 10):
    if num % 2 == 0:
        print "Found an even number", num
        continue
    print "Found an odd number", num
```

The default for the code is to always print an odd number was found, but the continue statement jumps back to the top of the loop.

R. Jesse Chaney



CS344 – Oregon State University

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

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

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



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.



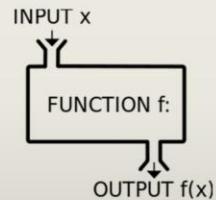
Defining Functions

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

Defining a new function
is pretty straightforward
in Python. Note the colon.

```
add(4, 3)
```

Calling your new
function is also easy.



- Function declaration doesn't specify return type
- All functions return a value (the *None* value if not specified)
- Parameter data types are not specified

R. Jesse Chaney

CS344 – Oregon State University

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)
```



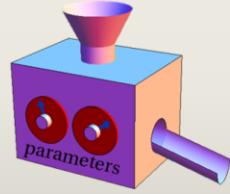
Defining Functions, cont

```
def factorial_iter(x):  
    ans = 1  
    for i in range(2, x + 1):  
        ans = ans * i  
    return ans
```

An iterative function definition.

```
def factorial_recur(x):  
    if x == 0:  
        return 1  
    else:  
        return x * factorial_recur(x - 1)
```

A recursive function definition.



R. Jesse Chaney

CS344 – Oregon State University

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.



Defining Functions, cont



```
def factorial_iter(x = None): # A default value for the
    if x is None:             function, if one is not supplied.
        print "you must supply an argument value"
        return 0
    ans = 1
    for i in range(2, x + 1):
        ans = ans * i
    return ans
```

R. Jesse Chaney

CS344 – Oregon State University

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()
```



Importing Modules

```
import subprocess # spawn new processes
import os         # portable way of using OS dependent functionality
import stat       # functions for interpreting the results of os.stat()
import time       # various time-related functions
```

A module is a Python file that (generally) has only definitions of variables, functions, and classes. It is like a library of functions and values.



R. Jesse Chaney

CS344 – Oregon State University

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 and 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
```



input() vs raw_input()



<code>var = raw_input("prompt string")</code>	returns whatever the user typed, up to hitting the ENTER key, as a string
<code>var = input("prompt string")</code>	returns the evaluation as a numeric type of the keystrokes typed

R. Jesse Chaney

CS344 – Oregon State University

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.

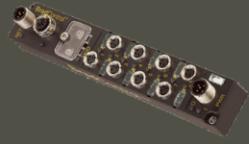
If you want to format the string a bit more, you can use the *str.rjust(#)* and *str.ljust(#)* methods. You specify the preferred width of the output field. If the argument happens to be wider than the given width, the output is not truncated.

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.



File I/O



```
f = open(file_name [, access_mode][, buffering])
f.close()
```

- f.read([size]) - reads some quantity of data and returns it as a string.
- f.readline() - reads a single line from the file; a newline character (\n) is left at the end of the string.
- f.readlines() - returns a list containing all the lines of data in the file.
- f.write(str) - writes the contents of string to the file, returning *None*. It does not add a newline character to the end of the string. The string may be either text or binary data.

R. Jesse Chaney

CS344 – Oregon State University

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..."
```



“Pythonic” Coding Style



- Use 4-space indentation, and **no (hard) tabs**.
- Wrap lines so that they don't exceed 79 characters.
- Use blank lines to separate functions and classes, and larger blocks of code inside functions.
- When possible, put comments on a line of their own.
- Use docstrings.
- Use spaces around operators and after commas, but not directly inside bracketing constructs.
- Name your classes and functions consistently; the convention is to use CamelCase for classes and lower_case_with_underscores for functions and methods.

R. Jesse Chaney

CS344 – Oregon State University

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 Code

```
python project1_prob1.py
```

This will identify a variable called `__name__`
with a value of “`__main__`”



R. Jesse Chaney

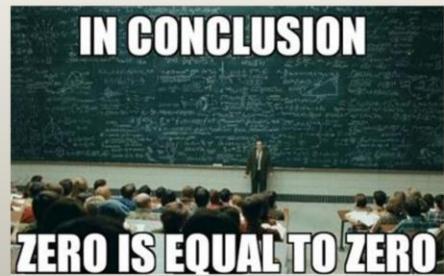
CS344 – Oregon State University

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



Conclusions

- History of Python
- Some of the Major users of Python
- Python tutorial sections 1-7



R. Jesse Chaney

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

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