Let´s Learn Python!

What is programming?

- A **computer** is a machine that **stores** and **manipulates** information

- A **program** is a detailed set of **instructions** telling a computer exactly what to do.

Instructions for people:

"Clean your room."
- my mom, circa 1992

"Mail your tax return no later than April 15th."
- the IRS

"I'll have a burger with cheese, pickles and onions."
- me, at the drive-thru

Let’s talk to Python!

Types of Python

Python Types

CPython:

- It is implementation of Python in ‘C’

Jython:

- It is an integration of Python with Java

IronPython:

- It is an integration of .NET and C# with Python

PyObjC:

- It is an integration of Python with Objective C

Python Types

PyJS:

- It is implementation of Python with JavaScript

Finally,

Python Types

Comment

Comments

Single Line Comment:

- '#' is used for Single Line comment in Python

  #This is a comment

Multi Line Comment:

- You can use triple-quoted strings. When they're not a docstring they are ignored.

Python Pro Tip

Math

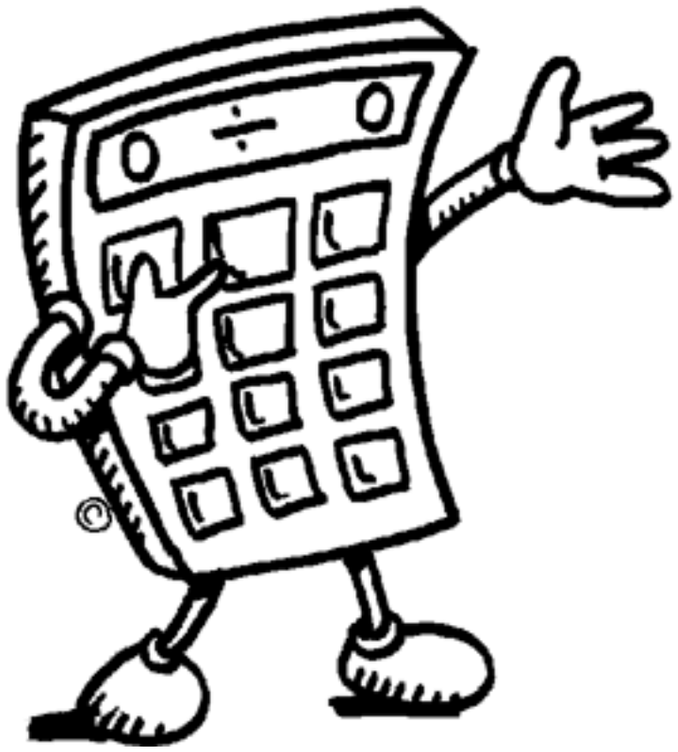
Math

Arithmetic operators:

addition: +

subtraction: -

multiplication: *



Try doing some math in the interpreter:

```
>>> 1 + 2
>>> 12 - 3
>>> 6 * 5
```

Math

Another arithmetic operator:

division: /

Try doing some division in the interpreter:

```
>>> 8 / 4
>>> 20 / 5
>>> 10 / 3
```

Is the last result what you expected?

Math

Rule:

★ If you want Python to respond in floats, you must talk to it in floats.

<u>Integers (whole numbers)</u>:

```
9
-55

>>> 11/3
3
```

<u>Floats (decimals)</u>:

```
17.318
10.0

>>> 11.0/3.0
3.6666666666666665
```

Math

Comparison operators:

==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Math

Comparison practice:

```
>>> 5 < 4 + 3
>>> 12 + 1 >= 12
>>> 16 * 2 == 32
>>> 16 != 16
>>> 5 >= 6
```

Guess the answer, then try in the Python shell.

Math

Comparison practice:

```
>>> 5 < 4 + 3          True
>>> 12 + 1 >= 12       True
>>> 16 * 2 == 32       True
>>> 16 != 16           False
>>> 5 >= 6             False
```

Strings

Strings

```
>>> “garlic breath”
>>> “Thanks for coming!”
```

Try typing one without quotes:

```
>>> apple
```

What’s the result?

If it’s a string, it must be in quotes.

```
>>> “apple”
>>> “What’s for lunch?”
>>> “3 + 5”
```

Strings

String operators:

concatenation (adding words together): +

multiplication: *

Try concatenating:

```
>>> “Hi” + “there!”
'Hithere!'
```

Try multiplying:

```
>>> “HAHA” * 250
```

Variables

Variables

Calculate a value:

```
>>> 12 * 12
144
```

How can you save that value, `144`?

Assign a <u>name</u> to a <u>value</u>:

```
>>> donuts = 12 * 12
>>> color = "yellow"
```

A **variable** is a way to *store a value*.

Variables

```
>>> donuts = 12 * 12
>>> color = “yellow”
```

Assign a <u>new</u> value:

```
>>> color = “red”
>>> donuts = 143

>>> color = “fish”
>>> color = 12
>>> color
12
```

Variables

- ★ Calculate once, keep the result to use later
- ★ Keep the name, change the value

Some other things we can do with variables:

```
>>> fruit = “watermelon”
>>> print fruit[2]
>>> number = 3
>>> print fruit[number-2]
```

Errors

Errors

```
>>> “friend” * 5
‘friendfriendfriendfriendfriend’

>>> “friend” + 5
Error

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

Do you remember what ‘concatenate’ means?
What do you think ‘str’ and ‘int’ mean?

Errors

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
    ^                       ^          ^          ^
```

- Strings: ‘str’
- Integers: ‘int’
- Both are objects
- Python cannot concatenate objects of different *types*

Errors

Here's how we would fix that error:

```
>>> "friend" + 5
Error
```

Concatenation won't work.

Let's use the `print` command for display:

```
>>> print "friend", 5
friend 5
```

No concatenation, no problem!

Types of data

Data types

Three types of data we already know about:

“Hi!”	string
27	integer
15.238	float

Python can tell us about types using the `type()` function:

```
>>> type(“Hi!”)
<type ‘str’>
```

Can you get Python to output `int` and `float` types?

Data type: Lists

Lists

List: a sequence of objects

```
>>> fruit = [“apple”, “banana”, “grape”]
>>> numbers = [3, 17, -4, 8.8, 1]
```

Guess what this will output:

```
>>> type(fruit)
```

```
>>> type(numbers)
```

Lists

List: a sequence of objects

```
>>> fruit = [“apple”, “banana”, “grape”]
>>> numbers = [3, 17, -4, 8.8, 1]
```

Guess what this will output:

```
>>> type(fruit)
<type ‘list’>

>>> type(numbers)
<type ‘list’>
```

Lists

Index: Where an item is in the list

```
>>> fruit = [“apple”, “banana”, “grape”]
>>> fruit[0]
'apple'
```

```
['apple', 'banana', 'grape']
    0          1          2
```

Python always starts at zero!

Lists

```
>>> print fruit[0]
apple

>>> fruit
['apple', 'banana', 'grape']
```

How would you use `type()` to verify the type of each element in the list?

Lists

Make a **list** of the three primary colors.

Use an **index** to print your favorite color's name.

Lists

Make a **list** of the three primary colors.

```
>>> colors = [‘red’, ‘blue’, ‘yellow’]
```

Use an **index** to print your favorite color’s name.

```
>>> print colors[1]
```

Data type: Booleans

Booleans

A boolean value can be: `True` or `False`.

Is 1 equal to 1?

```
>>> 1 == 1
True
```

Is 15 less than 5?

```
>>> 15 < 5
False
```

Booleans

What happens when we type Boolean values in the interpreter?

```
>>> True
>>> False
```

When the words ‘True’ and ‘False’ begin with capital letters, Python knows to treat them like Booleans and not strings or integers.

```
>>> true
>>> false
>>> type(True)
>>> type(“True”)
```

Booleans

Combine comparisons:

and: <u>All</u> must be correct to be True

```
1 == 1    and    2 == 2
True      and    True     --> True

>>> True and True
>>> True and False
>>> False and False
```

Booleans

Combine comparisons:

`or`: <u>Only one</u> must be correct to be True

```
1 == 1    or    2 != 2
True      or    False     --> True

>>> True or True
>>> False or True
>>> False or False
```

Booleans

Reverse a Boolean:

`not`: `True` **becomes** `False`
`False` **becomes** `True`

```
not 1 == 1   --> False
not True     --> False
```

Booleans: Practice

Try some of these expressions in your interpreter:

```
>>> True and True
>>> False and True
>>> 1 == 1 and 2 == 1
>>>"test" == "test"
>>> 1 == 1 or 2 != 1
>>> True and 1 == 1
>>> False and 0 != 0
>>> True or 1 == 1
>>>"test" == "testing"
>>> 1 != 0 and 2 == 1
```

Logic

`if` Statements

`if` Statements

Making decisions:

"**If** you're not busy, let's eat lunch now."
"**If** the trash is full, go empty it."

If a condition is met, perform the action that follows:

```
>>> name = "Jesse"
>>> if name == "Jesse":
        print "Hi Jesse!"

Hi Jesse!
```

`if` Statements

Adding more choices:

"**If** you're not busy, let's eat lunch now.
Or **else** we can eat in an hour."

"**If** there's mint ice cream, I'll have a scoop.
Or **else** I'll take butter pecan."

The `else` clause:

```
>>> if name == "Jesse":
        print "Hi Jesse!"
    else:
        print “Impostor!”
```

`if` Statements

Including many options:

"**If** you're not busy, let's eat lunch now.
Or **else if** Bob is free I will eat with Bob.
Or **else if** Judy's around we'll grab a bite.
Or **else** we can eat in an hour."

The `elif` clause:

```
>>> if name == "Jess":
        print "Hi Jess!"
    elif name == “Sara”:
        print “Hi Sara!”
    else:
        print “Who are you?!?”
```

`if` Statements

`if/elif/else` practice

Write an if statement that prints "Yay!" if the variable called color is equal to "yellow".

Add an elif clause and an else clause to print two different messages under other circumstances.

Loops

Loops

Loops are chunks of code
that repeat a task over and over again.

★*Counting* loops repeat a
certain number of times.

★*Conditional* loops keep
going until a certain thing happens
(or as long as some condition is True).

Loops

Counting loops repeat a certain number of times.

```
>>> for mynum in [1, 2, 3, 4, 5]:
        print “Hello”, mynum

Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
```

The *for* keyword is used to create this kind of loop, so it is usually just called a *for loop*.

Loops

Conditional loops repeat until something happens.

```
>>> count = 0
>>> while (count < 4):
        print 'The count is:', count
        count = count + 1

The count is: 0
The count is: 1
The count is: 2
The count is: 3
```

The *while* keyword is used to create this kind of loop, so it is usually just called a *while loop*.

Functions

Functions

Remember our PB&J example?

Which is easier?:

1. Get bread
2. Get knife
4. Open PB
3. Put PB on knife
4. Spread PB on bread ...

1. Make PB&J

Functions are a way to *group* instructions.

Functions

What it's like in our minds:

"Make a peanut butter and jelly sandwich."

In Python, it could be expressed as:

```
make_pbj()
```

function **name**

Functions

Let's *create* a function in the interpreter:

```
>>> def say_hello():
        print 'Hello'
```

The second line should be indented 4 spaces.

Hit enter until you see the prompt again.

Functions

Now we'll *call* the function:

```
>>> say_hello()
Hello
```

Functions

What if we wanted to make many kinds of sandwiches?

"Make a peanut butter and jelly sandwich."
"Make a cheese and mustard sandwich."

In Python, it could be expressed as:

```
make_pbj(bread, pb, jam)
make_pbj(bread, cheese, mustard)
```

function **name** function **parameters**

Functions

Let's *create* a function with parameters in the interpreter:

```
>>> def say_hello(name):
        print 'Hello', name
>>> say_hello("Katie")
Hello Katie
```

Functions: Practice

1. Work alone or with a neighbor to create a function that **doubles a number** and prints it out.

Functions: Practice

1. Work alone or with a neighbor to create a function that **doubles a number** and prints it out.

```
>>> def double_number(number):
        print number * 2

>>> double_number(14)
28
```

Functions: Practice

2. Work alone or with a neighbor to create a function that takes **two numbers**, multiplies them together, and prints out the result.

Functions: Practice

2. Work alone or with a neighbor to create a function that takes **two numbers**, multiplies them together, and prints out the result.

```
>>> def multiply(num1, num2):
        print num1 * num2

>>> multiply(4, 5)
20
```

Functions: Output

`print` displays something to the screen.

But what if you want to save the value that results from a calculation, like your doubled number?

```
>>> new_number = double_number(12)
>>> new_number
24
```

Functions: Output

```
>>> def double_number(number):
...     return number * 2

>>> new_number = double_number(12)
24

>>> new_number
```

Functions

Rules:

- ★ Functions are **defined** using **`def`**.
- ★ Functions are **called** using **parentheses**.
- ★ Functions take **parameters** and can return **outputs**.
- ★ `print` **displays** information, but does not give a value
- ★ `return` gives a **value** to the caller (you!)

Input

Input

Input is information that we enter into a function so that we can do something with it.

```
>>> def hello_there(name):
        print "Hello there", name
>>> hello_there("Katie")
"Hello there Katie"
```

But what if you want to enter a different name?
Or let another user enter a name?

Input

The `raw_input()` function takes *input* from the user - you give that input to the function by typing it.

```
>>> def hello_there():
        print “Type your name:”
        name = raw_input()
        print “Hi”, name, “how are you?”
```

Input

```
>>> def hello_there():
        print “Type your name:”
        name = raw_input()
        print “Hi”, name, “how are you?”


>>> hello_there()
Type your name:
Barbara
Hi Barbara how are you?
```

Input

A shortcut:

```
>>> def hello_there():
        name = raw_input("Type your name: ")
        print "Hi", name, "how are you?"

>>> hello_there()
Type your name: Barbara
Hi Barbara how are you?
```

Objects

Objects

Real objects in the real world have:

- things that you can do to them (actions)
- things that describe them (attributes or properties)

In Python:

- "things you can do" to an object are called *methods*
- "things that describe" an object are called *attributes*

Objects

This ball object might have these *attributes*:

```
myBall.color
myBall.size
myBall.weight
```

You can display them:

```
print myBall.size
```

You can assign values to them:

```
myBall.color = 'green'
```

You can assign them to attributes in other objects:

```
anotherBall.color = myBall.color
```

Objects

The ball object might have these *methods*:

```
ball.kick()
ball.throw()
ball.inflate()
```

Methods are the things you can *do* with an object.

Methods are chunks of code - *functions* - that are included *inside* the object.

Objects

In Python the description or blueprint of an object is called a *class*.

```
class Ball:

  color = 'red'
  size = 'small'
  direction = ''

  def bounce(self):
    if self.direction == 'down':
      self.direction == 'up'
```

Objects

Creating an instance of an object:

```
>>> myBall = Ball()
```

Give this instance some attributes:

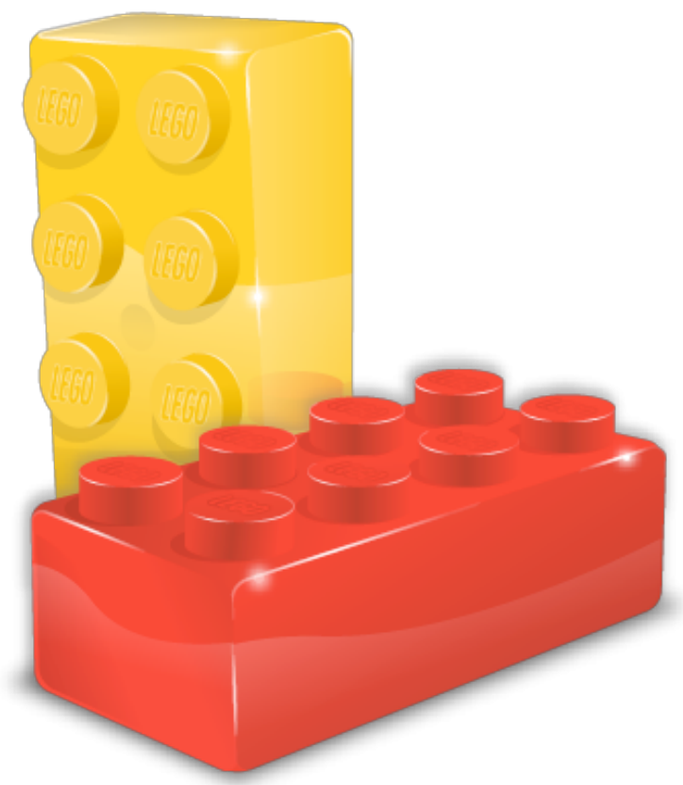
```
>>> myBall.direction = “down”
>>> myBall.color = “blue”
>>> myBall.size = “small”
```

Now let’s try out one of the methods:

```
>>> myBall.bounce()
```

Modules

Modules



A module is a block of code that can be combined with other blocks to build a program.

You can use different combinations of modules to do different jobs, just like you can combine the same LEGO blocks in many different ways.

Modules

There are lots of modules that are a part of the Python Standard Library

How to use a module:

```
>>> import random
>>> print random.randint(1, 100)
>>> print random.choice([1, 2, 3])

>>> import time
>>> time.time()
```

Modules

Some more!

```
>>> import datetime
>>> datetime.now()

>>> import calendar
>>> calendar.prmonth(2013, 3)
>>> calendar.pryear(1980)
```

That’s it!

Modules

Turtles!

```
>>> import turtle
>>> turtle.reset()
>>> turtle.forward(20)
>>> turtle.right(20)
>>> turtle.forward(20)
>>> turtle.bye()
```

You can find out about other modules at: http://docs.python.org