

# Let's Learn Python!

Young Coders at PyCon 2016

# Meet your teachers:

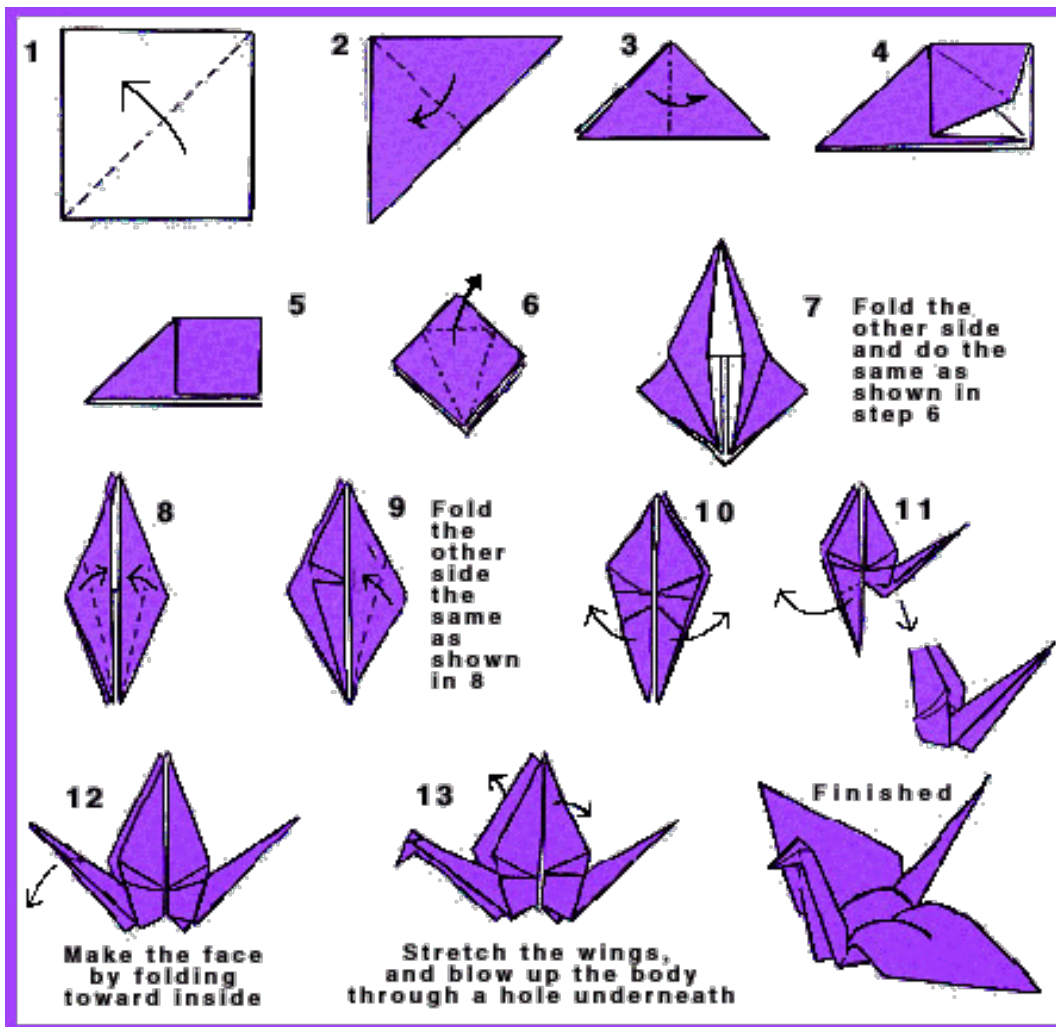
Katie Cunningham  
Barbara Shaurette



# What is programming?

- A problem to solve
- A solution to the problem
- The solution translated into a language the computer can understand

- ★ A computer is a machine that stores pieces of information.
- ★ A computer also moves, arranges, and controls that information (or *data*).
- ★ A program is a detailed set of instructions that tells a computer what to do with data.



## SUPER-EASY BIND

NO GLUE, NO SCISSORS, NO NEEDLE, NO RULES



## Sugar Cookies

Preheat oven to 375 degrees

10 minutes

### Ingredients:

2/3 cup margarine

3/4 cup sugar

1 egg

1/2 teaspoon vanilla

2 cups flour

1 1/2 teaspoons baking powder

1/4 teaspoon salt

4 teaspoon milk

### Instructions

Combine margarine, sugar, egg, and vanilla. Separately, combine flour, baking powder, and salt. Combine two mixtures along with milk.

Chill 1 hour before baking

Roll out dough to about a 1/4 of an inch thickness and cut out cookies.

# Algorithms

# 97 Simple Steps to a PB&J

Is making PB&J difficult?

How many steps does it feel like?

Let's talk to Python!



# Math

Try doing some math at the prompt:

>>> 1 + 2

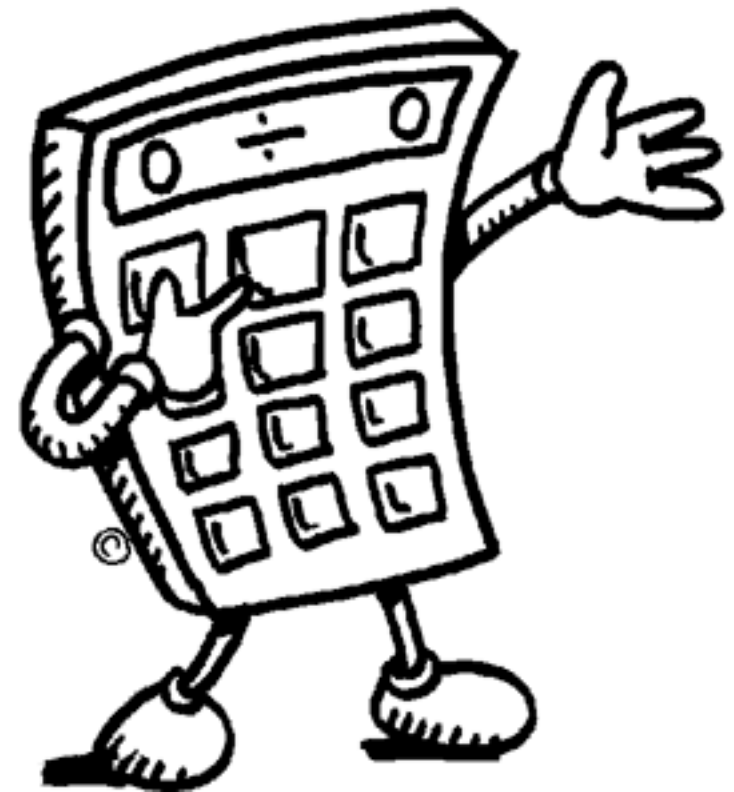
>>> 12 - 3

>>> 9 + 5 - 15

Operators:

add: +

subtract: -



# Math

More operators:

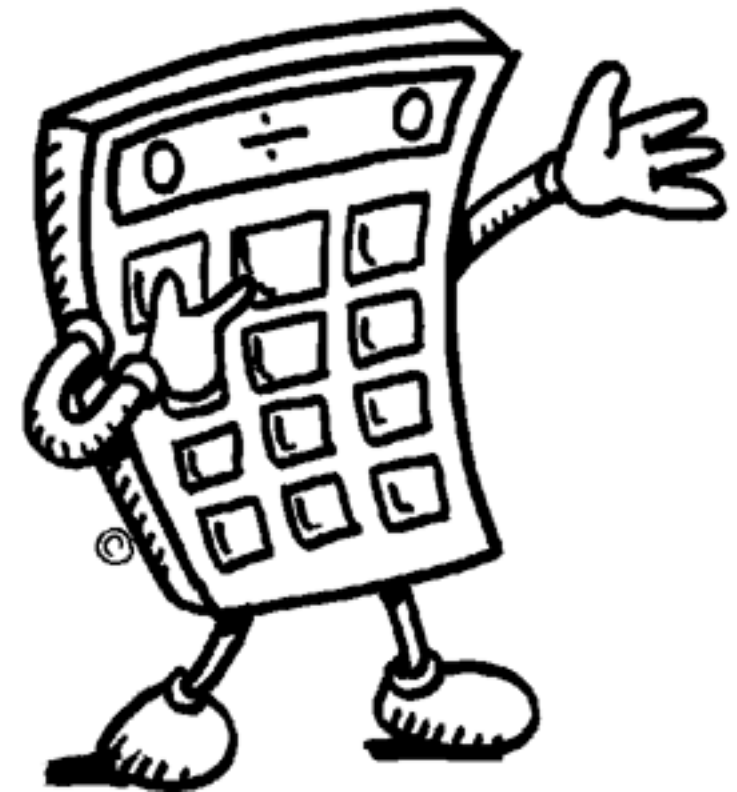
divide: /

multiply: \*

```
>>> 6 * 5
```

```
>>> 6 / 2
```

```
>>> 10 * 5 * 3
```



# Math

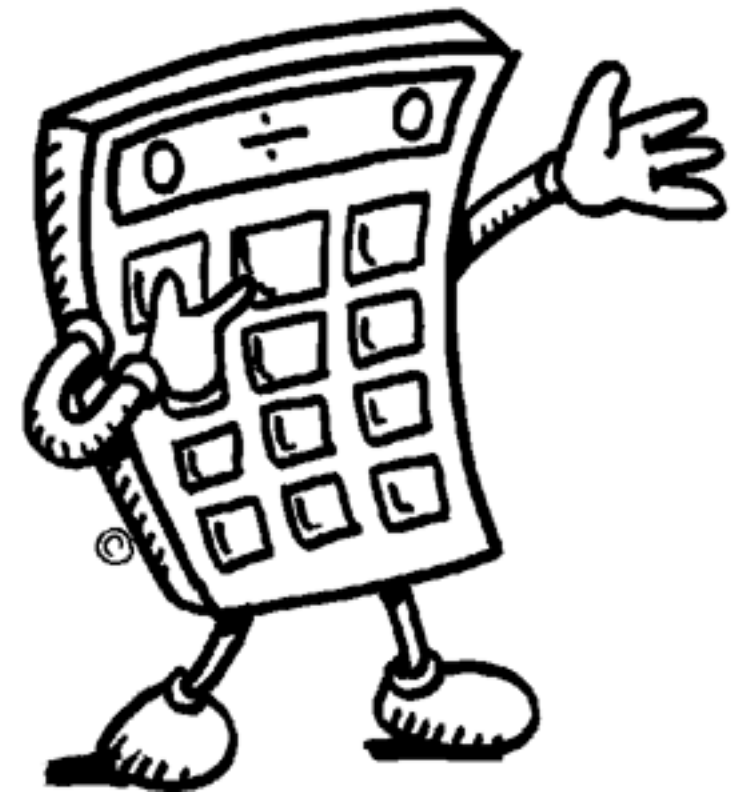
Try some more division:

>>> 8 / 4

>>> 20 / 7

>>> 10 / 3

Are you getting the  
results you expected?



# Math

**Floats (decimals):**

**10.0**

**17.31**

```
>>> 10/3
```

```
3.3333333333333335
```

```
>>> 10/2
```

```
5.0
```

**Integers:**

**9,**

**-55**

```
>>> round(10/3)
```

```
3
```

```
>>> round(10/2)
```

```
5
```

# Math

## Comparison operators:

<code>==</code>	Equal to
<code>!=</code>	Not equal to
<code>&lt;</code>	Less than
<code>&gt;</code>	Greater than
<code>&lt;=</code>	Less than or equal to
<code>&gt;=</code>	Greater than or equal to

# Math

## Practice:

>>> 5 < 4 + 3

>>> 12 + 1 >= 12

>>> 16 \* 2 == 32

>>> 16 != 16

>>> 5 >= 6

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

# Math

## Practice:

```
>>> 5 < 4 + 3
```

```
True
```

```
>>> 12 + 1 >= 12
```

```
True
```

```
>>> 16 * 2 == 32
```

```
True
```

```
>>> 16 != 16
```

```
False
```

```
>>> 5 >= 6
```

```
False
```

# Strings



# Strings

Examples:

```
>>> "garlic breath"  
>>> "Hello!"
```

Try typing one without quotes:

What's the result?

```
>>> apple
```



Rule:

A string must be in quotes

```
>>> "apple"  
>>> "What's for lunch?"  
>>> "3 + 5"
```

# Strings

String operators:

concatenation: +

multiplication: \*

Try concatenating: `>>> "Hi" + "there!"`

Try multiplying: `>>> "HAHA" * 250`

# Strings: Indexes

Strings are made up of **characters**:

```
>>> "H" + "e" + "l" + "l" + "o"  
'Hello'
```

Each character has a position called an *index*:

H	e	l	l	o
0	1	2	3	4

In Python, indexes start at **0**

# Strings: Indexes

```
>>> print("Hello"[0])  
H
```

```
>>> print("Hello"[4])
```

```
>>> print("Hey, Bob!"[4])
```

```
>>> print("Hey, Bob!"[6-1])
```

# Strings: Indexes

## Rules:

- ★ Each character's position is called its *index*.
- ★ Indexes start at 0.
- ★ Spaces inside the string are counted.

# Variables

# Variables

Calculate a value:      `>>> 12 * 12`  
                                 `144`

How can you save that value?

Give the value a name:      `>>> donuts = 12 * 12`  
                                 `>>> donuts`  
                                 `144`

# Variables

Create a variable  
and give it a value:

```
>>> color = "yellow"  
>>> color  
'yellow'
```

Now assign a  
new value:

```
>>> color = "red"  
>>> color  
'red'
```

```
>>> color = "fish"  
>>> color = 12
```



# Variables

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

# Variables

Some other things we can do with variables:

Math operations

```
>>> donuts = 12 * 12
>>> fishes = 3
>>> fishes + donuts
```

String operations

```
>>> color = "yellow"
>>> day = "Monday"
>>> color + day
>>> color * fishes
>>> color + day * fishes
```

# Variables

More things we can do with variables:

Get an index  
from a string:

```
>>> fruit = "watermelon"  
>>> print (fruit[2])
```

Do some math  
to get the index:

```
>>> mynumber = 3  
>>> print (fruit[mynumber-2])
```

# Variables

Assigning values or making comparisons?

```
>>> fruit = "watermelon"
```

```
>>> 5 = 6
```

```
>>> fruit == "watermelon"
```

```
>>> 5 == 6
```

# Errors

# Errors

```
>>> "friend" * 5  
'friendfriendfriendfriendfriend'
```

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

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: Can't convert 'int' object to str implicitly
```

What do you think 'str' and 'int' mean here?

Does this error message tell you what's wrong?

# Errors

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

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: Can't convert 'int' object to str implicitly
```

- 'int' is an integer
- 'str' is a string
- Python cannot concatenate objects of different *types*

# Errors

How can we fix this error?  
Concatenation won't work.

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

What if we make 5 a string?

```
>>> "friend" + "5"  
friend5
```

What's another way that we  
could fix this error?

Let's do something new with  
the `print` command:

```
>>> print("friend", 5)  
friend 5
```



# Types of data

# Data types

We already know about three types of data:

"Hi!"	string
27	integer
15.238	float

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

```
>>> type("Hi!")  
<class 'str'>
```

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

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 *upper case*  
letters, Python knows to  
treat them like Booleans  
instead of strings or integers.

```
>>> true  
>>> false  
>>> type(True)  
>>> type("True")
```

# Booleans

**and**

If both are True: `>>> 1==1 and 2==2`  
True

If only one is True: `>>> 1==1 and 2==3`  
False

If both are False: `>>> 1==2 and 2==3`  
False

# Booleans

or

If both are True:  $>>> 1==1 \text{ or } 2==2$   
True

If only one is True:  $>>> 1==1 \text{ or } 2!=2$   
True

If both are False:  $>>> 1==2 \text{ or } 2==3$   
False

# Booleans

## not

You can use the word  
**not** to reverse the  
answer that Python gives:

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

```
>>> not 1==1  
False
```

Any expression that is  
True can become False:

```
>>> not True  
False
```



# Booleans

You can also use booleans in their own expressions:

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

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

```
>>> not True and True  
>>> not True or True
```

# Booleans: Practice

Try some of these expressions in your interpreter.

See if you can predict what answers Python will give back.

```
>>> 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" == "tests"
>>> 1 != 0 and 2 == 1
```

# 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)  
<class 'list'>
```

```
>>> type(numbers)  
<class '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

Make a **list** of three of your favorite colors.

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

# Lists

Make a **list** of three of your favorite colors.

```
>>> colors = ["red", "orange", "purple"]
```

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

```
>>> print(colors[1])
```



Logic

# if Statements

# if Statements

Making decisions: "If you're hungry, let's eat lunch."

"If the trash is full, go empty it."

If a condition is met,  
perform an action:

```
>>> name = "Katie"  
>>> if name == "Katie":  
    print("Hi Katie!")
```

Hi Katie!

# if Statements

Adding a choice:

"If you're hungry, let's eat lunch.  
Or else we can eat in an hour."

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

Adding a choice in  
our code with the  
*else clause*:

```
>>> if name == "Katie":  
        print("Hi Katie!")  
    else:  
        print("Impostor!")
```

# if Statements

Adding many  
choices:

"If there's mint ice cream, I'll have a scoop.  
Or else if we have vanilla, I'll have 2!  
Or else if there's chocolate, give me 3!  
Or I'll just have a donut."

Adding more  
choices in our code  
with the *elif* clause:

```
>>> if name == "Katie":  
        print("Hi Katie!")  
    elif name == "Barbara":  
        print("Hi Barbara!")  
    else:  
        print("Who are you?")
```

# if Statements

## if/elif/else practice

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

Add an *elif* clause and an *else* clause to print two different messages for other values of the variable.

(Here's our last example)

```
>>> name = "Katie"
>>> if name == "Katie":
        print("Hi Katie!")
    elif name == "Barbara":
        print("Hi Barbara!")
    else:
        print("Who are you?")
```

# if Statements

## if/elif/else practice

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

Add an *elif* clause and an *else* clause to print two different messages for other values of the variable.

```
>>> color = "blue"
>>> if color == "yellow":
    print("Yay!")
    elif color == "purple":
    print("Try again!")
else:
    print("We want yellow!")
```

# 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 - they keep going until they get to the end of a count.

```
>>> for mynum in [1, 2, 3, 4, 5]:  
        print("Hello", mynum)
```

```
Hello 1
```

```
Hello 2
```

```
Hello 3
```

```
Hello 4
```

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 (or as long as some condition is True).

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

```
The count is: 0
```

```
The count is: 1
```

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 looks easier?:

1. Get bread
2. Get knife
4. Open peanut butter
3. Put peanut butter on knife
4. Spread peanut butter on bread
5. ...

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, you could say it like this:

```
make_pbj(bread, pb, jam, knife)
```



function name

function parameters

# 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_sandwich(bread, filling, toppings)
```



*function name*



-----



-----



*function parameters*

# Functions

Let's define a function in the interpreter:

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

Now we'll call the function:

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



# Functions

Let's define a function with parameters:

```
>>> def say_hello(myname):  
        print("Hello", myname)
```

```
>>> say_hello("Katie")  
Hello Katie
```

```
>>> say_hello("Barbara")  
Hello Barbara
```

# Functions

A few things to know about functions ...

```
>>> def say_hello(myname):  
        print("Hello", myname)
```

<code>def</code>	This is a <b>keyword</b>	We use this to let Python know that we're defining a function.
<code>myname</code>	This is a <b>parameter</b> (and a <b>variable</b> ).	We use this to represent values in the function.
<code>print(...)</code>	This is the <b>body</b>	This is where we say what the function <i>does</i> .

# Functions: Practice

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

(Here's our last example)

```
>>> def say_hello(myname):  
        print("Hello", myname)
```

# Functions: Practice

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

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

```
>>> double(14)  
28
```

```
>>> double("hello")  
hellohello
```

# 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.

(Here's our last example)

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

```
>>> double(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.

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

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

```
>>> multiply("hello", 5)  
hellohellohellohellohello
```

# Functions: Output

`print` displays something to the screen.

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

We call the function, passing it the number 12:

```
>>> double(12)  
24
```

We call the function again, passing it the number 12 and assigning it to the variable `new_number`:

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

But what happens here?

```
>>> new_number
```

# Functions: Output

This time let's use `return`  
instead of `print`.

```
>>> def double(number):  
      return number * 2
```

We call the function,  
passing it the number 12:

```
>>> double(12)  
24
```

We call the function again,  
passing it the number 12  
assigning the value to the  
variable `new_number`:

```
>>> new_number = double(12)  
  
>>> new_number
```

Now what happens?



# Functions

- ★ 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 (that's you!)

Input

# Input

**Input** is information we pass to a function so that we can do something with it.

In this example, the string “Katie” is the input, represented by the variable myname.

```
>>> def hello_there(myname):  
        print("Hello", myname)
```

```
>>> hello_there("Katie")  
'Hello there Katie'
```

# Input

The `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 = input()  
    print("Hi", name, "how are you?")
```

# Input

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

```
>>> hello_there()
```

Type your name: Barbara

Hi Barbara how are you?

# 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

Lots of modules are included in the Python Standard Library.  
Here's how you can use a few of these modules:

Generate a random  
number between 1-100:

```
>>> import random  
>>> random.randint(1,100)
```

What timezone does your  
computer think it's in?:

```
>>> import time  
>>> time.tzname
```

Print a calendar for this  
month!:

```
>>> import calendar  
>>> calendar.prmonth(2016, 6)
```

# Modules

Print the names  
of all the files in  
a directory:

```
>>> import os  
>>> for file in os.listdir("/home/pi"):  
    print(file)
```

Open a web  
page and read 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>

Let's make a game!

# Games!

Open a new window (File > New File) and type these lines:

```
secret_number = 7

guess = input("What number am I thinking of? ")

if secret_number == guess:
    print("Yay! You got it.")
else:
    print("No, that's not it.")
```

Choose Run > Run Module. Save to Desktop as 'guess.py'.

# Games!

Open a new window (File > New File) and type these lines:

```
from random import randint  
secret_number = randint(1, 10)  
while True:  
    guess = input("What number am I thinking of? ")  
  
    if secret_number == guess:  
        print("Yay! You got it.")  
        break  
    else:  
        print("No, that's not it.")
```

Choose Run > Run Module. Save to Desktop as 'guess2.py'.

# Games!

Open a new window (File > New File) and type these lines:

```
from random import randint

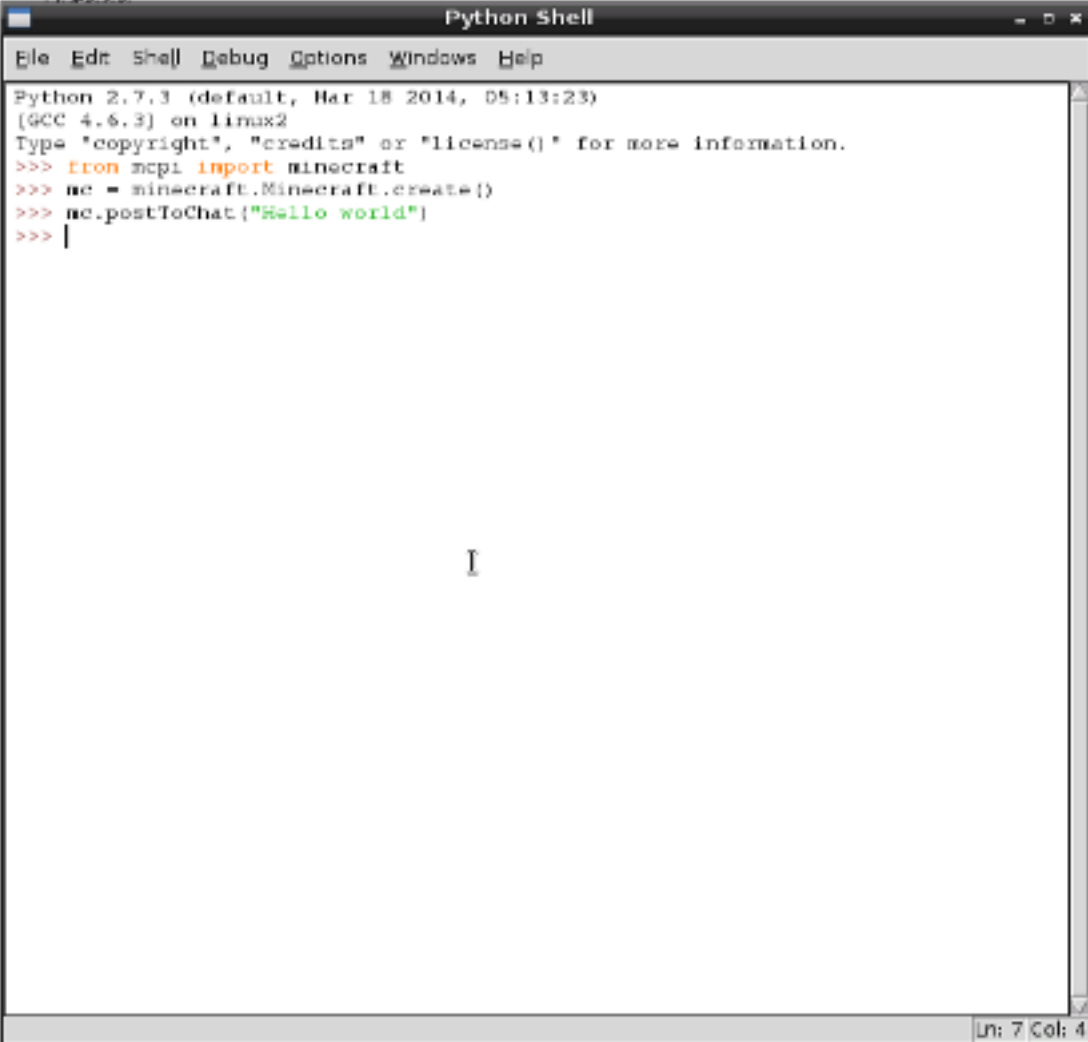
secret_number = randint(1, 10)

while True:
    guess = input("What number am I thinking of? ")

    if secret_number == guess:
        print("Yay! You got it.")
        break
    elif secret_number > guess:
        print("No, that's too low.")
    else:
        print("No, that's too high.")
```

Choose Run > Run Module. Save to Desktop as 'guess3.py'.

# Minecraft!



A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The text area shows the following code and output:

```
Python 2.7.3 (default, Mar 18 2014, 05:13:23)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> from mcpi import minecraft
>>> mc = minecraft.Minecraft.create()
>>> mc.postToChat("Hello world")
>>> |
```

The status bar at the bottom right indicates "Ln: 7 Col: 4".





# Minecraft!

```
>>> from mcpi import minecraft
```

```
>>> mc = minecraft.Minecraft.create()
```

```
>>> mc.postToChat("Hello world")
```

```
>>> pos = mc.player.getPos()
```

```
>>> print pos.x, pos.y, pos.z
```

```
>>> mc.player.setPos(pos.x, pos.y+100, pos.z)
```

```
>>> mc.setBlock(pos.x+1, pos.y, pos.z, 1)
```

*(Air: 0, Grass: 2, Dirt: 3)*

# Minecraft!

```
>>> from mcpi import block
```

```
>>> dirt = block.DIRT.id
```

```
>>> mc.setBlock(x, y, z, dirt)
```

```
>>> stone = block.STONE.id
```

```
>>> mc.setBlocks(x+1, y+1, z+1, x+11, y+11, z+11, stone)
```

```
>>> tnt = 46
```

```
>>> mc.setBlocks(x+1, y+1, z+1, x+11, y+11, z+11, tnt)
```

```
>>> mc.setBlocks(x+1, y+1, z+1, x+11, y+11, z+11, tnt, 1)
```

Congratulations!

You're now a Pythonista!

## *What can YOU do with Python?*

- Make more games
- Edit music and videos
- Build web sites
- Write a program that does your homework for you ...
- What are some other ideas?

# Raspberry Pi

Help setting up your new computer:

<http://www.raspberrypi.org/quick-start-guide>

Minecraft on your new Raspberry Pi:

<https://www.raspberrypi.org/learning/getting-started-with-minecraft-pi/worksheet/>

<http://www.stuffaboutcode.com/p/minecraft-api-reference.html>