

Welcome to Programming Bootcamp!

Make sure you sign in!

Course philosophy

- The lecture is the least important part.
- I can't teach you everything, so I will teach you a few things very well.
- Course goal: get you comfortable enough with Python basics that you can easily learn more on your own when the course is done.

Advice for getting the most out of this course

- **Do the problem sets!!**
- Review the answers when they are released
- Always try to solve problems yourself first
- BUT: don't be afraid to ask questions when you get stuck!

Conventions used in these slides

- Text written in `fixed width font` represents actual Python code or terminal commands.
- I will color certain pieces of code in different colors for clarity. For example:
 - code comments will be `green`
 - reserved words and built-in functions will be `blue`

4

NOTE: If you try to copy and paste code from this powerpoint, be aware that any code containing curly quotes (`"""` `"""`) will cause errors. You need to change them to regular straight quotes first (just retype them in your text editor or interpreter and they will be corrected).

“comments” “reserved words” and “built-in functions” will be explained later

Introduction to Programming and Python

Programming Bootcamp 2015

Day 1 – 6/2/15

Today's topics

1. What is a program?
2. Writing and running Python code
3. Print statements
4. Error messages
5. Variables & data types
6. Basic math

Note: If you're following along online, see the notes panel (below the slides) for some additional clarifying information.

6

Hello, this is the notes panel 😊

1. What is a program?

What is a program?

Program - A set of instructions that tells the computer how to perform a task.

The words “program” and “script” can often be used interchangeably, though “script” is generally used to describe quicker, simpler programs.

8

You can think of a program a bit like a recipe: it gives step-by-step instructions on how to combine data and functions (ingredients) to achieve the desired end result (a tasty cake)

The program must be written in a language the computer can understand. This is where languages such as Python come in.

What is a programming language?

- Computers only understand binary (0's and 1's)
- Writing code in binary is very hard for humans.
- So, programming languages were created:
 - Human writes code in human-friendly language
 - An “interpreter” or “compiler” translates the code into computer language
 - The computer runs the translated code
- Python is just one of many such “human-friendly” programming languages.

Why learn Python?

- It's a particularly simple and easy to learn language
- It's widely used by scientists
- Personal preference

2. Writing and running Python code

Two main ways to use Python

- Interactively: execute one line of code at a time using the interpreter
 - allows you to immediately see the result of each line of code
 - good for quickly testing small pieces of code
- Using scripts: write code in a file and then execute the file
 - allows you to easily re-run the same code
 - allows you to share your code with others

12

There are actually other ways to write and run Python code that meld these two models. For example, an IPython notebook lets you see the result of a line or block of code immediately, while still saving all your code in an editable and shareable format. I won't have time to go over this, but it's worth checking out (it works particularly well for exploratory data analysis).

Using the interpreter

```
Sarah@Russet: ~/Dropbox/Python/PythonBootcamp2013/Tab1
$ python
Python 2.7.3 (default, Dec 18 2012, 13:50:09)
[GCC 4.8.3] on cygwin
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

An example of the Python interpreter initiated from within the terminal.

- The Python interpreter is an interactive terminal environment for running Python code.
- When you type a line of code in the interpreter, it is **immediately** executed (in contrast to a script, which is only executed when you tell it to).
- Although you could theoretically write full programs in the interpreter, it would not be ideal. Instead, we'll be using the interpreter as a way of **quickly testing small bits of code**.

Using the interpreter (do this now)

- Open your terminal.
- Type `python`
- You should see your command prompt change to:
`>>>`
- You can now type any Python code and see the result immediately. Try typing the following and hit enter:

```
print "Hello world!"
```

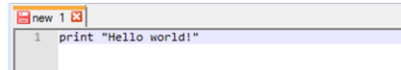
```
>>> print "Hello world!"  
Hello world!  
>>> |
```

- You can exit this environment by typing `quit()`

Creating & running a Python script

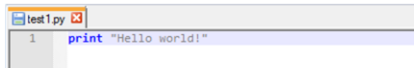
Step 1: Creating a script

- Open a plain text editor (Notepad++, TextWrangler)
- Type the following:



```
new 1
1 print "Hello world!"
```

- Save your file in your lab1 folder as `test1.py`
- *Note:* Depending on your text editor, you may notice some of the code has changed colors. This is called syntax highlighting:



```
test1.py
1 print "Hello world!"
```

15

Side notes:

- Syntax highlighting makes reading code easier by color-coding different types parts of your code (e.g. functions = blue, strings = gray, numbers = red)
- A good text editor has syntax highlighting specific to a variety of different programming languages
- The editor determines what language you're using based on the file extension you save your file as – in this case, `.py` indicates Python.

Creating & running a Python script

Step 2: Running the script

- Open your terminal and navigate to the folder where you saved your script (use `cd`, `ls/dir`, and `pwd`).
- Once in the correct folder, type:
`python test1.py`
- Python will now attempt to execute your script. If there are no errors in your code, you should see something like this:

```
ssh@herbert: ~/Dropbox/Python/PythonBootcamp2013/Tab1
$ python test1.py
Hello world!
```

16

You always have to either be in the same directory as your script, or provide a path to the script when running it.

An example of providing a path would be something like this:

```
python ../../my_code/test1.py
```

This would of course change depending on which directory you're in relative to the script.

So which method should I use?

- For most actual code-writing, you will definitely want to write a script
- I will use the interpreter for many examples in these lectures to show what each line of code does
- I will also ask you to use the interpreter for certain quick exercises

3. Print statements

Print statements

`print` `"Hello world!"`

command text
aka *"function"* aka *"string"*

- This is called a *print statement*.
- The `print` command prints text (or other content) to the terminal screen.
- In order to print a line of text (called a *string* in the programming world), we must enclose the text in quotes. Note that the quotes were not printed to the screen.

19

Side notes:

- The `print` command works somewhat differently than other commands we'll be seeing in the future
- Most commands/functions are in the format `command(arg)`
- In fact, the `print` command can be written this way too—try `print("Hello world")`. (This is optional for `print`, but not for other commands)

Test out printing (do this now)

- Try the following in the interpreter:

```
print "Hello" + "world"  
print "Hello", "world"  
print 123  
print 1 + 2  
print 2 * 3  
print "1 + 2"  
print "1 + 2 =", 1 + 2, "!"
```

Tip: you can press the up arrow to cycle through your code history

20

Things to note:

- Yes, you can do math this way, and it works pretty much the way you'd expect
- Numbers do not require quotes around them. If you do add quotes, the numbers will be treated as *strings* rather than numbers.
- A comma can be used to print multiple things at once. In particular, this is necessary for when you would like to print a mix of strings and numbers (e.g. the last example)
- Note that using a comma in a print statement automatically adds a space in the printed output in place of the comma

Variations on the print statement

Code	Output	Reason
<code>print "hello world"</code>	hello world	The text in quotes is printed as typed (the quotes are not printed)
<code>print "hello" + "world"</code>	helloworld	"Adding" strings together concatenates them
<code>print "hello"</code> <code>print "world"</code>	hello world	A "newline" (return) is inserted after each print statement
<code>print "hello",</code> <code>print "world"</code>	hello world	Adding the comma at the end of the print statement suppresses the automatic newline
<code>print "hello", "world"</code>	hello world	adding a comma also allows you to print multiple things in one print statement. (this will be useful when we start using variables.) Notice that a space was automatically inserted between the words
<code>print "hello" * 3</code>	hellohellohello	Print something multiple times.
<code>print 5</code>	5	You can print numbers (no quotes)
<code>print 1 + 2</code>	3	You can also print the results of math operations

21

Don't worry about trying to remember all this right now, just know that these sorts of things are possible and refer back to this slide when you need it.

More on valid math operations later in this lecture.

4. Understanding error messages

Understanding error messages

Type the following in your interpreter:

```
print hello
```

Understanding error messages

Type the following in your interpreter:

```
print hello
```

Output:

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'hello' is not defined
```


Understanding error messages

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'hello' is not defined
```

This is a typical Python error message. It gives you several pieces of useful information to help you figure out what went wrong:

1. **A line number** - This is the line in your program where the error occurred. If you are using a script, the first thing you should do is find this line and examine it.
2. **An explanation** - This message should give you some idea of the nature of the error.

Sometimes these messages are hard to understand. If you can't figure out what's wrong, your best bet is to copy and paste the “explanation” part of the error message into Google. Look for forum posts of other people who got your same error.

25

Side notes:

- Don't expect to find someone with exactly the same problem as you when you do a search. Just skim the answers for general hints of the type of problems that give this error, and see if they apply to you.
- Sometimes just searching the name of the error, in this case “NameError”, will give better results. You can try searching “python NameError” to make sure your results are specific to python
- Not all error messages will look exactly like this, but most should have some useful info you can use.

Understanding error messages

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'hello' is not defined
```

So what went wrong here?

- Basically, we forgot to put quotes around `hello`. When Python sees a word with no quotes around it, it assumes that word is a *variable*.
- Since we did not create a variable called `hello` yet, Python displays an error message.
- If we had been running a script, Python would have terminated the execution of the script as soon as the error was encountered.

5. Variables and data types

Introduction to variables

What is a variable?

- You can think of variables as little boxes that we put data in. You name each box so that you can refer to it and use it in your code. This gives your code flexibility (for reasons you will see soon).
- Creating a variable is sometimes called *declaring* or *defining* a variable. This needs to be done before you can use the variable.
- You can name your variables almost anything, but avoid using names that are also commands in Python (e.g. don't name a variable "print")

Introduction to variables

For example:

```
geneID = "uc007ndd.1"
```

Introduction to variables

For example:

```
geneID = "uc007ndd.1"
```

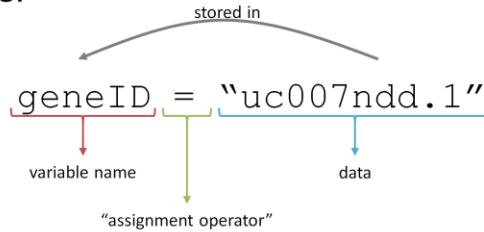
variable name

"assignment operator"

data

Introduction to variables

For example:



In english: Store the value "uc007ndd.1" in "geneID"

31

Side notes:

- Although the equals sign is used here, don't think of this as an equation. A single equals sign in Python means that we are *assigning* the value on the right side to the variable on the right side.
- The values of a variable can be overwritten by simply assigning something new to it. For example, if in this example `geneID` already had a value, we would have overwritten it here by saying `geneID = "uc007ndd.1"`

Important: Choosing variable names

What words are ok to use for variable names?

- **Rules** (if you break these, you'll get errors):
 - only letters, numbers, and underscores can be used in a variable name
 - the variable name can not begin with a number
 - you can not use any of the python reserved words as a variable name (see table at end of slides)
 - the capitalization of your variables matters. For example, `geneID` and `geneid` would be considered different variables.
- **Conventions** (recommended):
 - begin a variable name with a lower case letter
 - use a name that is descriptive of the info stored in the variable
 - if your variable name is more than one word squished together, use camelCase or under_scores to make it easier to read.

32

Side notes:

- if it's not clear, "camel case" means leaving the first word lower case and all following words upper case to make it easier to read. `justLikeThis`.
- Is it ok to capitalize the first letter of you variable names? Yes, but it's almost never done, because in some language capitalized words are used for a different naming purpose (naming classes, to be specific).

Examples of variable names

Good variable names:

- `geneID`
- `personCount`
- `input_file`
- `avgGeneCount`

Bad variable names:

- `3rdColumn` (illegal)
- `sdasqweksf` (gibberish)
- `person#` (illegal)
- `class` (reserved word)

Python reserved words

The following words have special meaning to python and can not be used as variable names:

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with
yield	True	False	None		

If you get a weird error for a variable, check if it's on this list.

34

If you have proper syntax highlighting in your text editor, it should be obvious when you accidentally use a reserved word because it will be a different color than all your other variables!

Examples

What will this code print?

```
geneID = "uc007ndd.1"  
print geneID
```

35

Will there be quotes around it?

Examples

What will this code print?

```
geneID = "uc007ndd.1"  
print geneID
```

Result

```
uc007ndd.1
```

Examples

What will this code print?

```
apples = 5  
oranges = 10  
fruit = apples + oranges  
print fruit
```

Examples

What will this code print?

```
apples = 5  
oranges = 10  
fruit = apples + oranges  
print fruit
```

Result

15

38

We can add two variables together if they both contain numbers.

Examples

What will this code print?

```
apples = 5  
oranges = 10  
print apples + oranges
```

Examples

What will this code print?

```
apples = 5  
oranges = 10  
print apples + oranges
```

Result

15

40

Note that the addition is done before printing.

Examples

What will this code print?

```
apples = 5  
oranges = 10  
print apples, oranges
```

Examples

What will this code print?

```
apples = 5  
oranges = 10  
print apples, oranges
```

Result

```
5 10
```

42

So the comma is not the same as the +. The comma is basically for separating multiple types of things we would like to print.

Examples

What will this code print?

```
apples = 5  
oranges = 10  
print "I have", apples, "apples"
```

Examples

What will this code print?

```
apples = 5  
oranges = 10  
print "I have", apples, "apples"
```

Result

```
I have 5 apples
```

44

This allows us to insert the value of a variable into our print statement.

Examples

What will this code print?

```
people = 3  
people = people + 1  
print people
```

Examples

What will this code print?

```
people = 3  
people = people + 1  
print people
```

Result

4

46

This is an important one to understand. What we did here was overwrite the value of *people* with the value of *people + 1*. Since the right side of the = sign is evaluated first, the value of the right side is $3 + 1$, or 4. This is then assigned to the variable on the left, which in this case overwrites the value of *people* with the number 4.

Later on, this is how we will create counters, i.e. variables that we increment by 1 whenever we do a certain task.

Examples

What will this code print?

```
people = 3  
animals = 4  
people = animals  
print people  
print animals
```

Examples

What will this code print?

```
people = 3  
animals = 4  
people = animals  
print people  
print animals
```

Result

```
4  
4
```

48

Another example of overwriting a variable. Note that the value of animals is unchanged.

Examples

What will this code print?

```
name = "Joe Shmo"  
age = 20  
print name, "will be", (age + 1), "next year"
```

Examples

What will this code print?

```
name = "Joe Shmo"  
age = 20  
print name, "will be", (age + 1), "next year"
```

Result

```
Joe Shmo will be 21 next year
```

50

Here we did some math within the *print* statement.

If we now printed the value of *age*, what would you expect it to be?

=> Answer: it would still be 20! We did not overwrite the value of *age*. We simply added 1 to its value when we printed. To overwrite a variable, we must use the assignment operator (equals sign) with the variable on the left hand side.

Examples

What will this code print?

```
yourAge = "16"  
print "You will be", (yourAge + 1), "next year"
```

Examples

What will this code print?

```
yourAge = "16"  
print "You will be", (yourAge + 1), "next year"
```

Result

```
You will be  
Traceback (most recent call last):  
  File "L2_variables.py", line 2, in <module>  
    print "You will be", (yourAge + 1), "next year"  
TypeError: cannot concatenate 'str' and 'int' objects
```

Examples

What will this code print?

```
yourAge = "16"  
print "You will be", (yourAge + 1), "next year"
```

Result

```
You will be  
Traceback (most recent call last):  
  File "L2 variables.py", line 2, in <module>  
    print "You will be", (yourAge + 1), "next year"  
TypeError: cannot concatenate 'str' and 'int' objects
```

What happened?

- We put the number 16 in quotes--this makes it a string instead of an integer!
- Python can't do addition with strings, so it gives an error message.
- Notice that it starts to print the message, but fails when we try to do the addition.

53

Note, Python *can* actually do addition of two string, e.g. "16" + "1". However, the result of this will be string addition (aka *concatenation*), not numerical addition. The result will therefore be "161" instead of 17.

Variable data types

- Data comes in different types – numbers, words, letters, etc.
- In Python, certain types of data are treated differently. There are four main “**data types**” we’ll be working with:
 1. **String** - a *string* is just another word for text. You can think of it as "a string of letters/characters". Strings are enclosed in double or single quotes to distinguish them from variables and commands (ex: "This is a string!" 'So is this!')
 2. **Integer ("int")** - this refers to whole numbers (same as in real life). In programming, integers are handled differently than non-integers, which is why we make this distinction.
 3. **Floating point numbers ("float")** - numbers with decimals.
 4. **Booleans** – True or False (1 or 0). We’ll talk more about this later.

Converting between data types

- As we've seen, different types of data are treated differently by Python:

```
>>> print 1 + 1      } No quotes; treated as integers → sum  
2  
>>> print "1" + "1" } Quotes; treated as strings → concatenate  
11  
>>> print 1 + "1"   } Mixed; not compatible! → error
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: unsupported operand type(s) for +: 'int'  
and 'str'
```

55

Converting between data types

- Sometimes we'll want to convert one data type into another.
- Python provides simple built-in functions for a few types of conversions.
- Here's a partial list:

Function	Conversion type	Works on...
<code>str()</code>	Converts variable into a string	Integers, floats, booleans (and others)
<code>int()</code>	Converts variable into an integer	Numbers in string form Floats (decimal will be truncated) Booleans (True → 1, False → 0)
<code>float()</code>	Converts variable into a float (decimal)	Numbers in string form Integers (.0 will be added) Booleans (True → 1.0, False → 0.0)

Examples of conversion

```
>>> print 1 + int("1")
```

```
2
```

```
>>> print "1" + str(1)
```

```
11
```

```
>>> print float("1")
```

```
1.0
```

Now this works!

6. Basic math operations

Doing math in Python

Math in Python uses most of the symbols and conventions you're already used to:

```
>>> print 2 + 2
```

```
4
```

```
>>> print 5 * 5
```

```
25
```

```
>>> print 2 + 5 * 5
```

```
27
```

```
>>> print (2 + 5) * 5
```

```
35
```

} Order of operations
(P.E.M.D.A.S.) is maintained

Doing math in Python

There are a few differences, however:

<pre>>>> print 5 ** 2</pre>	}	Use ** for exponents
<pre>25</pre>		
<pre>>>> print 5 % 2</pre>	}	Use % to get remainders (aka "modulus" or "mod")
<pre>1</pre>		

Doing math in Python

The most important thing to watch out for is **integer division**:

```
>>> print 6 / 2
3
>>> print 5 / 2
2
>>> print 5 / 3
1
>>> print 5 / 4
1
```

Why are all of these answers rounded down?

61

6 / 2 works, but why are the rest rounded down?

Integer division

- Whenever you divide two integers, python always returns an integer answer
- Since integers are always whole numbers, python just truncates off the decimal
- To get a proper answer, **at least one of the numbers being divided must be a float:**

```
>>> print 5.0 / 2
2.5
>>> print float(5) / 2
2.5
```

*This is a very common source of errors, so keep it in mind when you divide!
When in doubt, convert one number with float().*

62

List of math operators

Operation	Result
$x + y$	sum of x and y
$x - y$	difference of x and y
$x * y$	product of x and y
x / y	quotient of x and y
$x // y$	(floored) quotient of x and y
$x \% y$	remainder of x / y
$x ** y$	x to the power y