Programming with Python

labs. Please make use of these resources.

Topical Syllabus

Why a "topical" syllabus?

Each group of students is unique and so, in different classes, we need to spend more or less time on specific topics. We will cover the topics below, more or less in the order indicated. I will be letting you know in class what to prepare for the next few classes.

- Introduction to computers and computing
- The Python environment
- Values and data types
- Input and output
- Hello World!
- Variables, and assignment
- Control Structures (if, if/else, while, for)
- Functions and modules
- Sequences, lists, strings, sets, tuples, dictionaries and comprehensions
- Working with text

- Files
- Classes – Object Oriented Programming (if time permits)
- additional material in-between and after the above, as time permits

Requirements

Exams

A midterm and final exam

Texts

Listed on the class website.

https://venus.cs.qc.cuny.edu/~waxman/cs90/
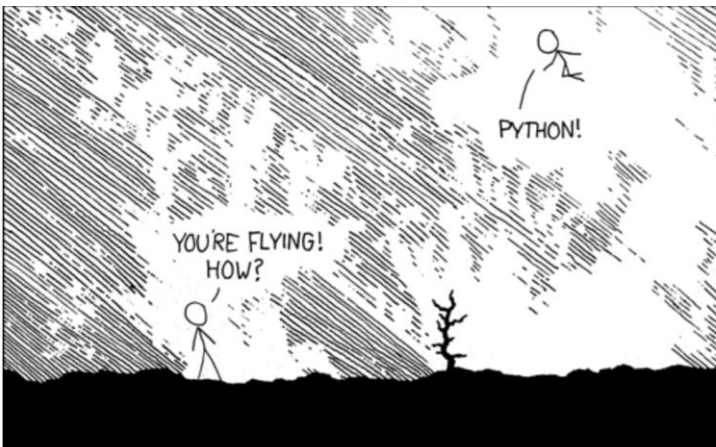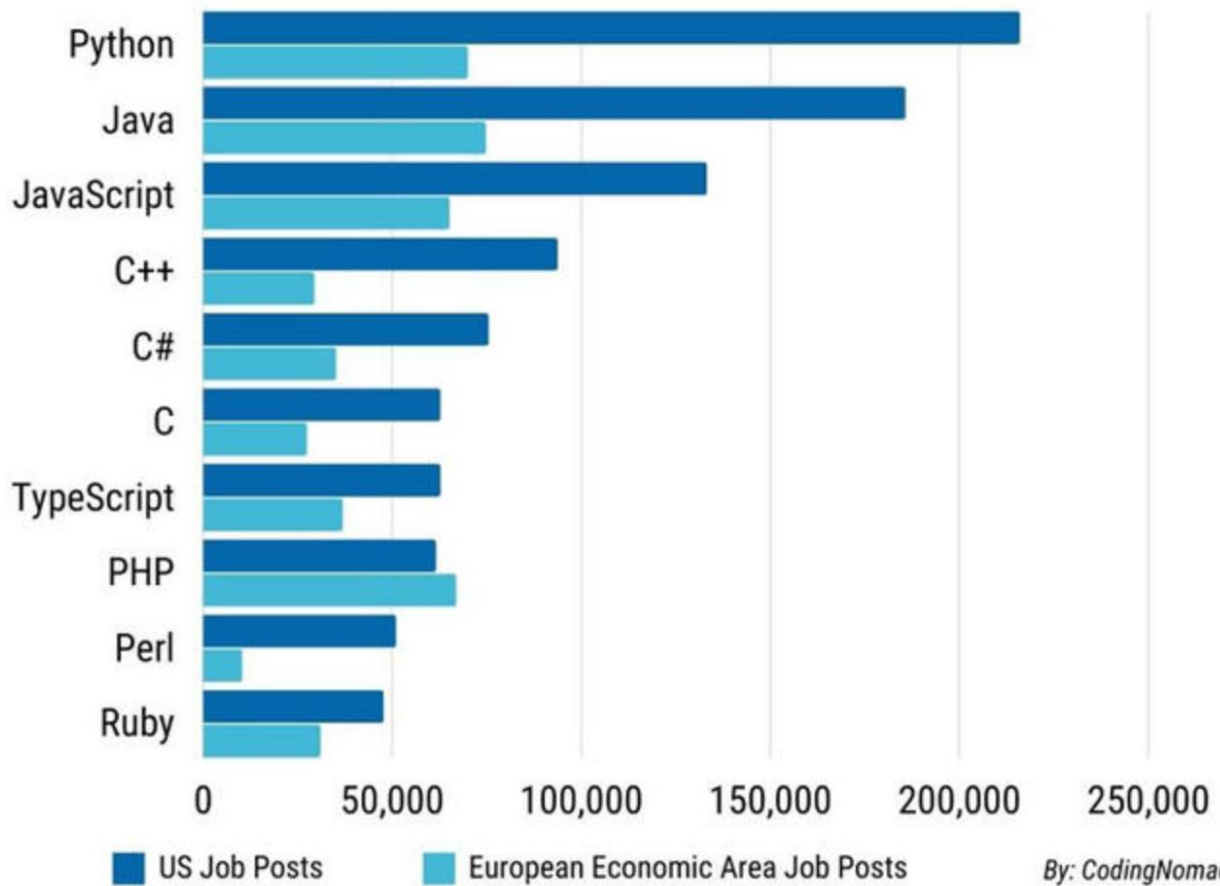
How to use these notes?

These notes are **not** a substitute for a text book. The notes will provide an <u>outline</u> of important ideas, and a place to work out the problems that we tackle in class. They contain most (but not necessarily all) of the material that we cover in the lecture.

There will be lots of space for you to write answers to question that come up in class. **Use these notes as your notebook to take class notes**.

Important: You are responsible for material that we cover in class but are not in the notes. If you miss any classes, please make sure that you are current.

Most in-demand programming languages of 2022

Based on LinkedIn job postings in the USA & Europe

Bar chart showing US Job Posts (dark blue) and European Economic Area Job Posts (light blue) for programming languages:
- Python
- Java
- JavaScript
- C++
- C#
- C
- TypeScript
- PHP
- Perl
- Ruby

Axis: 0, 50,000, 100,000, 150,000, 200,000, 250,000

US Job Posts — European Economic Area Job Posts — By: CodingNomads

import antigravity

PYTHON!

YOU'RE FLYING! HOW?

I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE! HELLO WORLD IS JUST print "Hello, world!"

I DUNNO... DYNAMIC TYPING? WHITESPACE? ... COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE! BUT HOW ARE YOU FLYING?

I JUST TYPED import antigravity THAT'S IT? ... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON. BUT I THINK THIS IS THE PYTHON.

Python
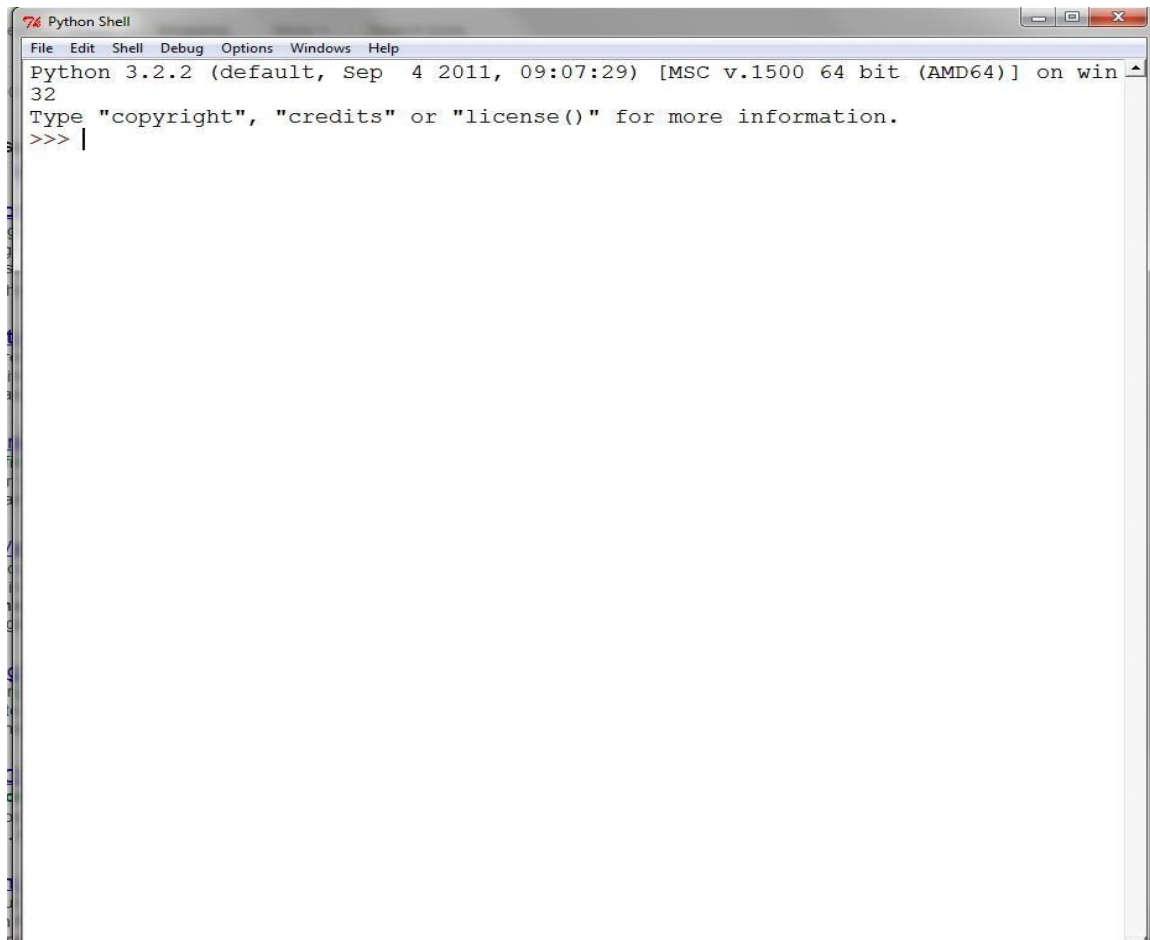
<u>Download Python and install it on your laptop.</u>

Python can be downloaded for free from the official Python web site:

<u>http://python.org/</u>

We will be using Python 3.6.4 (or whatever the current latest version is). Make sure you download and install the correct version (Windows/Mac) and the 32 or 64 bit version. Help is available at the NYU computer labs.

<u>IDLE – Integrated Development Environment</u>

When Python installs you will be able to access its functionality through the IDLE interface.

```
Python 3.2.2 (default, Sep  4 2011, 09:07:29) [MSC v.1500 64 bit (AMD64)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

We will be accessing Python in three different ways.

1. The interactive shell. This is available when we bring up IDLE. We will make limited use of this, mostly to experiment with various Python constructs and to test out ideas.

2. Python programs. This will be our main focus. A program (sometimes called a "script") is a file containing a sequence of statements in the Python language. We will create these files using IDLE and then "run" the file which will execute the statements in the program.

3. A Python Emulator. This website will allow us to "step through a python program and "see" what is happening in the memory after each instruction is executed. We will use this in class and it is an excellent tool when you are debugging (eliminating errors) your programs. It is available here: http://pythontutor.com/

And now …….

We first examine some Python constructs by using the interactive shell.
Data Types – A Classification of Python's Nouns

More precisely, a data type is thought of as the <u>combination</u> of the nouns and the verbs that they respond to.

Type Category	Type Name	Description
None	type(None)	The null object None
Numbers	int	Integer
	long	Arbitrary-precision integer (Python 2 only)
	float	Floating point
	complex	Complex number
	bool	Boolean (True or False)
Sequences	str	Character string
	unicode	Unicode character string (Python 2 only)
	list	List
	tuple	Tuple
	xrange	A range of integers created by xrange() (In Python 3, it is called range.)
Mapping	dict	Dictionary
Sets	set	Mutable set
	frozenset	Immutable set

We will start with some of the <u>basic</u> data types:

- Integers
- Floats
- Strings
- Booleans
.
There will be more later on.

The data types denote sets of the different types of Pythons nouns. **<u>The individual elements of the set, the constants, are the nouns</u>**. For example, One of Python's data types is "integer", i.e. all of the whole numbers. Each <u>particular number</u> is like a noun. It's like in a natural language there is the class of "nouns" (the data type) and individual nouns in that class.

Since one of the main things that we might want to do with Python is mathematical computations, Python provides two kinds of arithmetic "nouns", i.e. numbers, the integers and floats, and operations on them.

When we perform operations on the data we are generation something called an **<u>"expression."</u>** Simply put <u>an expression is something that Python can **evaluate,**</u> that is we can perform the operations and get a value. We will see examples as we look at the specific data types.

Important: In Python, everything is an (first class) object.

What does that mean in practice?

The first data type:

Integer: a whole number, positive, negative or 0. <u>Integers in Python can be arbitrarily long</u> and are written without commas. Other programming languages generally limit the size of integers.
0

56

 -897

3456789876543234567876655430937648300364554893022002

Note: we can use '_' as a separator to help visualize a long number. This will not effect its representation.

Ex: 123_456_789

Operations on integers:

Python provides the <u>standard mathematical operations</u> on integers, with a twist or two.

1. <u>Addition</u> ➔   123 + 56
Notice that adding two integers always produces another integer.

2. <u>Subtraction</u>➔ 123 – 56
Notice that subtracting two integers always produces another integer.

3. <u>Multiplication</u> ➔ 123*56
Notice that multiplying two integers always produces another integer.
4. <u>Division</u> There are **two division operations** available for integers: "/" and "//".

The "**/**" <u>operator</u>.

"/" will perform "regular" division so 123/56 ➔ **2.19**, i.e. produces a "floating point" number, (a number with a decimal point) which we will see next.

Even when the numbers divide exactly, the **result of "/" will always be a float and contain a decimal point.**

2/1 ➔ **2.0**

The "**//**" <u>operator</u>.

123//56 ➔ 2 Notice that the decimal point and all digits to its right are gone! They have been "**truncated**." It's as if Python took the 2.19 above and "chopped off" the decimal point and everything to its right.

This "//" operation on integers will **always** produce **an integer result**.

5. "%" the remainder function, called **mod**. If a and b are integers then a%b returns (evaluates to) the remainder of a divided by b.

For example:

6. Exponentiation: **

For example: 5**3=125

Operator Precedence (from the word precede)

What is that?

How do we evaluate expressions that contain many operations?

In order of the operator precedence.

Operator	Description
lambda	Lambda Expression
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
in, not in	Membership tests
is, is not	Identity tests
<, <=, >, >=, !=, ==	Comparisons
|	Bitwise OR
^	Bitwise XOR
&	Bitwise AND
<<, >>	Shifts
+, -	Addition and subtraction
*, /, %	Multiplication, Division and Remainder
+x, -x	Positive, Negative
~x	Bitwise NOT
**	Exponentiation
x.attribute	Attribute reference
x[index]	Subscription
x[index:index]	Slicing
f(arguments ...)	Function call
(expressions, ...)	Binding or tuple display
[expressions, ...]	List display
{key:datum, ...}	Dictionary display
`expressions, ...`	String conversion

The order in the table above is from **lowest to highest precedence.**

For example:

Important: Just like in algebra, we can modify the order of evaluation by using parenthesis.

The second data type:

Float: A <u>floating point number</u> is one that contains a decimal point. We saw an example of this above with the "/" operation on integers. There is an important technical difference between integers and floats. As we saw integers can be arbitrarily long (subject to the limitations of your particular computer hardware) and operations that yield integers are always exact. This is not the case with floats. Unlike integers, floats have maximum and minimum sizes.

max=1.7976931348623157e+308
min=2.2250738585072014e-308
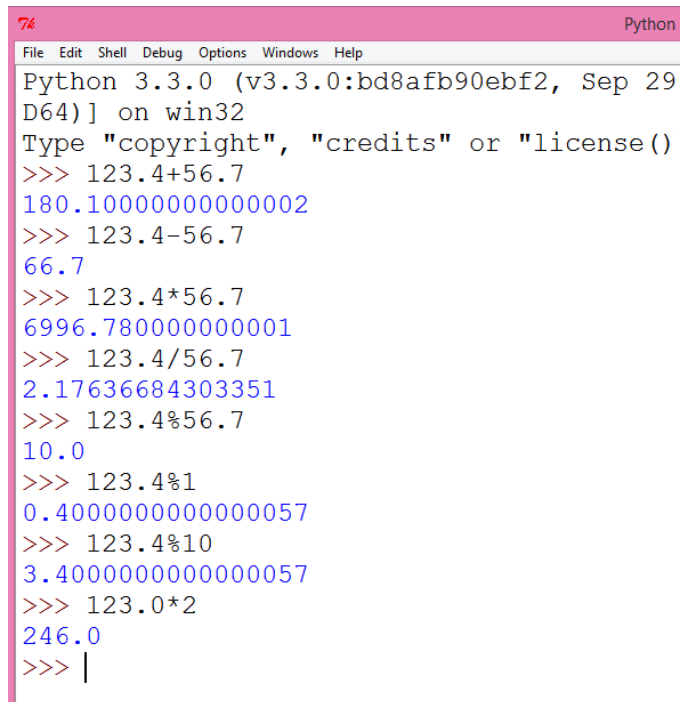
If any operation yields a float value larger than max, we have an **overflow** condition. Likewise, if it yields a value smaller than min, we have an **underflow** condition, and the values obtained are incorrect.

Additionally, because of how floats are represented in the computer, many values can only be approximated. Many floats are only approximate, $1/3 = 0.3333333333333333$.

Operations on floats:

Like it does for the integers, **Python provides the basic arithmetic operations** that you would expect. Python even defines the mod function for floats as the following examples illustrate.

```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29
D64)] on win32
Type "copyright", "credits" or "license()
>>> 123.4+56.7
180.10000000000002
>>> 123.4-56.7
66.7
>>> 123.4*56.7
6996.780000000001
>>> 123.4/56.7
2.17636684303351
>>> 123.4%56.7
10.0
>>> 123.4%1
0.4000000000000057
>>> 123.4%10
3.4000000000000057
>>> 123.0*2
246.0
>>>
```

Notice that operations involving two floats produce a float result. This is true even if one of the operand is an integer. So 123.0 * 2 produces a float even though 2 is an integer.

The next two data types are not arithmetic.

The third data type:

<u>String:</u> A string is a sequence of one or more **characters**. <u>An individual character is also a string</u> in Python. We denote a string by surrounding the character sequence by matching pairs of quotes. The quotes are: (1) ', (2)", (3) three of the first two – ''' or """. Here are some examples:

```
>>>
>>> 'hello'
'hello'
>>> "hello"
'hello'
>>> '''hello'''
'hello'
>>> """hello"""
'hello'
>>> 'hello"
SyntaxError: EOL while scanning string literal
```

Operations on strings:

Python provides a very rich set of string operations and functions. We will see them later on. For now,we look at one of the most important one**: "+" called concatenation.**

```
Python 3.3.2 (v3.3.2:d047928ae3:
tel)] on win32
Type "copyright", "credits" or
>>> 'Hello'+'there'
'Hellothere'
>>> 'Hello'+' here'
'Hello here'
>>> 'Hello'+' '+"everywhere"
'Hello everywhere'
>>>
```

What about 3*'Hello'? Can we "multiply strings?

```
>>> 3*'Hello'
'HelloHelloHello'
```

How come?????? … because just as 3*5 => 5+5+5 (repeated addition) likewise

3* 'Hello' => 'Hello'+ 'Hello'+ 'Hello'

Question: What about 0* 'Hello' ?

The forth data type:

Boolean: This data type has two values: True and False (spelled just as you see them – first letter caps and others lower case).

In addition … **certain operations yield Boolean values**, specifically, **comparison operations on arithmetic and string expressions yield comparison expressions.**

Arithmetic expressions are expressions that yield numbers. String expressions yield strings ('abc'+'def' ➔ 'abcdef'). We have seen these above.
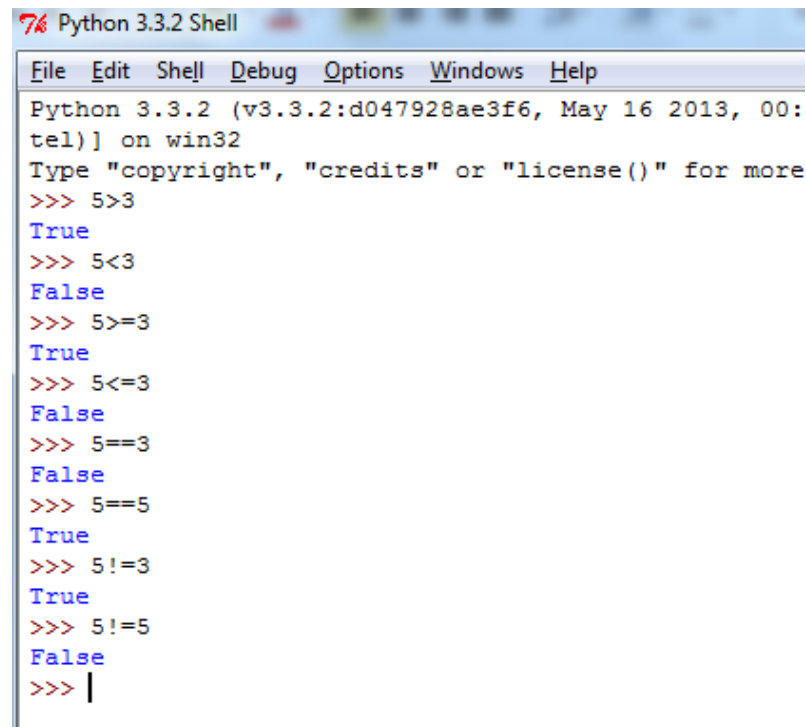
What are the **comparison operations provided by Python**?

Comparison operations

Python provides the following comparison operations:

```
>     greater than
<     less than
>=    greater than or equal to
<=    less than or equal to
==    equal to
!=    not equal to
```

When we use these to compare the values of arithmetic expressions**, we get a Boolean value**.

We can also use these comparison operations on strings. The value of the comparison depends of the lexicographic ordering (dictionary ordering – which string appears earlier or later in a dictionary) of the strings.

```
>>> 'abc'<'de'
True
>>> 'abc'<'ab'
False
>>> 'd'>'abc'
True
>>>
```

We can **chain the comparison operations in a standard "mathematical" way** and write things like:

x<=y<z where x,y,z are appropriate values.

Comparison expressions may be combined using **Boolean operations:**

and
or
not

The operator `not` yields `True` if its argument is false, `False` otherwise.

The expression `x and y` first evaluates *x*; if *x* is false, its value is returned; otherwise, *y* is evaluated and the resulting value is returned.

The expression `x or y` first evaluates *x*; if *x* is true, its value is returned; otherwise, *y* is evaluated and the resulting value is returned.

```
>>> True and True
True
>>> True and False
False
>>> False and False
False
>>> True or True
True
>>> True or False
True
>>> False or False
False
>>> True and 6
6
>>> True or 6
True
>>> False and 6
False
>>> False or 6
6
>>>
```

Look at this:

```
>>> (7>5)*6
6
>>> (7<5)*6
0
```

What is going on? How can we multiply (7>5) which evaluates to True by a number and get a number? Same question with (7<5).

Answer:

There are four more comparison operations provided by Python:

is
is not
in
not in

We will be using these later when we work with sequences and collections of various sorts.

Conversion between data types

Python provides a number of functions that let you convert between different data types:

float()

int()

str()

These are actually conversion constructors.

The function **type(x)** returns the "type', i.e. the kind of <u>object</u> x is.

```
>>> 5
5
>>> type(5)
<class 'int'>
>>> float(5)
5.0
>>> str(5)
'5'
>>> 5.0
5.0
>>> type(5.0)
<class 'float'>
>>> int(5.0)
5
>>> str(5.0)
'5.0'
>>> '5'
'5'
>>> type('5')
<class 'str'>
>>> int('5')
5
>>> float('5')
5.0
>>>
```

We can compare types:

```
>>> type(1)==type(1.0)
False
>>> type(5)==type('5')
False
>>> type(1)==type(5)
True
>>>
```

Output
How do we get Python to <u>output</u> information?

We have just seen the basic "nouns" (=data types) and some operations on them, i.e. item 3 below. We see that Python can get the computer to perform arithmetic and symbolic (e.g. string) manipulation.

Recall:

1. Input
2. Output
3. Memory
4. **Arithmetic – symbolic manipulation and evaluation**
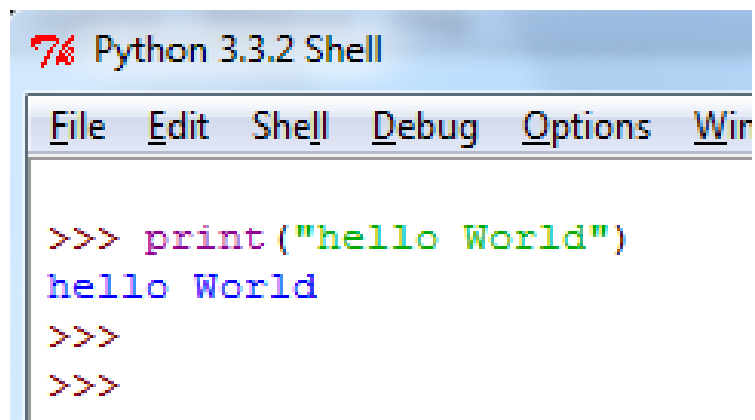5. Control

What about **output**?

1. Input
2. **Output**
3. Memory
4. Arithmetic – symbolic manipulation and evaluation
5. Control

Question: Where can we output to?

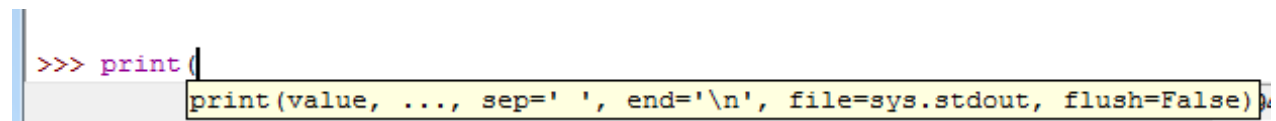For now, all of our output will be sent to the **monitor** (=screen).

How? We use the **print statement.**

print('Hello world')

```
76 Python 3.3.2 Shell
File  Edit  Shell  Debug  Options  Wir

>>> print("hello World")
hello World
>>>
>>>
```

Here is the syntax of the print statement.

```
>>> print(
         print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

As soon as I started to type the print command Python shows me the syntax. Here is how to read it.

First of all, **print is a "function."** A function has

- A name (here – **print** is the function name)
- Followed by parenthesis
- And zero or more "arguments" that are "passed in". If there is more than one argument, they are separated from each other by a comma.

15

What do the arguments mean/control?

value

sep

end

file

flush

<u>What is a default value:</u>

Now:

Print the values 1, 2, 3 all on one line

Print the values 1,2 3 one per line. Do this in two ways.

More examples using print: