Introduction to Programming

Introduction – What is this course about?

This course is an introduction to art and craft of programming using the Python programming language. We will assume no background in programming at all, but by the end of the semester, you should understand what programming is all about and should be quite proficient at composing interesting programs in Python.

Programming

This course is very “hands-on” because the only way to learn to program is by programming. We will frequently be working out problems together during the lecture, and there will generally be at least one program assigned for homework each week and at times … each class. Some will be quite straightforward, but some of the others, especially as the course progresses, … 😊 should be more fun …

Very important: please make sure that you set aside sufficient time to work on the programs.

You should first try to work the programs out by yourself. Sometimes this will be easy, sometimes a bit more difficult. You may discuss the projects with your classmates, and as the problems get more challenging that is a good idea, but the assignments you hand in must be your own work. Besides the college policy on plagiarism (and the academic consequences thereof, see: http://cs.nyu.edu/webapps/content/academic/undergrad/academic_integrity), what you gain from the course is directly proportional to what you put in.

Getting Help

Do not let things get out of hand! For many of you, the material in this course will be new and require a different way of thinking and, some of you, might need more time to “get it” than others. That is perfectly natural. But, please, do not be shy about asking for help when and if you need it. You can see me during my office hours (posted on the web site), write to the TA, and there are very knowledgeable tutors available in the computer 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
- Working with text
• additional material in-between and after the above, as time permits

Requirements

Exams

Two midterms (20% each) and a final (35%).

Projects

Twenty five percent (25 %) of the final grade.

How to hand in the homework.

Email the file to the TA at the email address on the web page. Given the size of the class we will be enforcing the homework deadlines strictly unless otherwise announced.

Texts

Listed on the class website.

http://cs.nyu.edu/courses/spring15/CSCI-UA.0002-006/index.html/

How to use these notes?

These notes are not a substitute for the text book. After a fairly detailed introduction, the notes will provide an outline of important ideas, and a place to work out the problems that we tackle in class. The 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 of the course will deal with **programming** in Python, but first a general introduction to computers and computing.

**What is a computer system? A “system” that does “computation.”**

So … what is required of such a system … and what is a “system” and what do we mean by “computation”?

Consider the following simple problem. We want this clever fellow to solve the problem on the paper.

**What** capabilities are required to solve it? **Where** are they “located?”

![Diagram of a stick figure with a box for input and output]
Where are the equivalent “capabilities” on a computer?

Front view:

Rear view:
Let’s look at the **hardware** of a typical computer system. It consists of:

1. **input devices:**
2. **output devices:**
3. **memory:** (“short term” and “long term”)

a. primary memory = **RAM** = Random Access Memory (for short term storage)

b. Hard drives, diskettes, CD ROM = **Compact Disk Read Only Memory**, USB (**Universal Serial Bus**) flash drives, DVD = **Digital Versatile Disc** or Digital Video Disc

**Aside: Binary numbers. Numbers expressed using only 1 and 0.**

---

Computers represent information in binary. Why?
Representing integers in binary

We usually write numbers in decimal notation. That means that each position represents a power of 10.

For example: 135 = one hundred and thirty five is "really"

\[ 135 = 1 \times 10^2 + 3 \times 10^1 + 5 \times 10^0 \]

"\times" in this context means multiply.

We can also express numbers in binary notation.

Huh?

Binary means that the "base" of the representation is 2 instead of 10. So here we are multiplying by powers of 2 rather than powers of 10.

Also whereas in the decimal number system there are 10 "digits" to choose from for each position, the numbers from 1 to 9, in binary there are only 2: 1 or 0.

Ex:

\[ 1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \]

What number is \(1101_2\) ?

Ex: Write out the binary numbers from 0 -7, using three "bits" (binary integers)

1

2

3

4

5

6

7
We can convert from decimal to binary notation and vice versa.

How? Here is an easy way.

Decimal to binary:

27

64

86

Binary to decimal

110

11100011
The following units are used to measure various “capacities” of storage devices.

Bit (b)

Byte (B)

KB

MB

GB

TB

PB

The anatomy of a floppy disk – ever see one?? Similar in some ways to a hard drive.

What happens when we format a diskette or the hard drive? (Remember formatting is a “destructive” operation)

….. but sometimes, not destructive enough. When? …

The anatomy of a hard drive.
c. ROM = **Read Only Memory**

4. System unit: *(Arithmetic Ability and Control)*

CPU = **Central Processing Unit**

ALU = **Arithmetic and Logic Unit**

Registers, etc. ....

Control unit
All connected by the **BUS. In the diagram below the BUS is represented by the double arrows.**

What about a modem? Does it represent …

Input?

Output?

Communication?
Modem = MO+ DEM

**MO**=Modulate  / **DEM**=DEModulate

AM radio = Amplitude **Modulation**
FM radio = Frequency **Modulation**

So the modem **Modulates** the carrier signal when it **sends it out**:

- it “adds” the binary information onto the analog carrier signal

and

**DeModulates** the signal when it is received

- It “subtracts out” the carrier signal when input is received and is left with the “bits”, i.e. binary digits (information) that hitched a ride on the carrier wave.

**Basic idea:**

The information is represented in the change from the underlying “carrier.” When we read words on a printed page, our brain “subtracts out” the blank “carrier” (white page) and the information is in the characters that changed the white background.
Python

In this course we will learn to program in the Python programming language. You can read something about its history in the textbook and on the web.

**Question:** There are there many programming languages: Python, C, C++, Java, Ruby, COBOL (still), Fortran (still too) ….. and many many more. **Why so many?**

**Answer:**

**Download Python and install it on your laptop.**

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


We will be using Python 3.3.3. 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.

**IDLE – Integrated Development Environment**

When Python installs you will be able to access its functionality through the IDLE interface.
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. Then we go on to programming.

Python is a programming language – i.e. a “language” that is used to program computers.

Just like a natural language, it contains

- nouns and
- verbs.

The nouns name “things” and the “verbs” tell what to do with the nouns

Throw ball!  ➔ verb noun

Eat apple!  ➔ verb noun

Read book. ➔ verb noun

Notice: different verbs for different nouns. You don’t eat a book or read an apple. It will be similar in Python.

**Data Types – A Classification of Python’s Nouns**

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

**The basic data types in Python are:**

- Integers
- Floats
- Strings
- Booleans

There will be more later on.

The data types denote sets of the different types of Pythons nouns. **The individual elements of the set, the constants, are the nouns.** For example, One of Python’s data types is “integer”, i.e. all of the whole numbers. Each particular number 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 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 **“expression.”** Simply put an expression is something that Python can evaluate, that is we can perform the operations and get a value. We will see examples as we look at the specific data types.
The first data type:

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

0
56
-897
345678987654323456787665543093764830036455489302002

Operations on integers:

Python provides the standard mathematical operations (we call them operators) on integers, with a twist or two.

1. **Addition**  \(123 + 56\)
   Notice that adding two integers always produces another integer.

2. **Subtraction**  \(123 - 56\)
   Notice that subtracting two integers always produces another integer.

3. **Multiplication**  \(123 \times 56\)
   Notice that multiplying two integers always produces another integer.

4. **Division**
   There are two division operations available for integers: “/” and “//”.

   The “/” operator.

   “/” will perform “regular” division so \(123/56 \Rightarrow 2.19\), 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.

   \(2/1 \Rightarrow 2.0\)

   The “//” operator.

   \(123//56 \Rightarrow 2\)  Notice that the decimal point and all digits to its right are gone! They have been “truncated.” Its 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:

```
Python Shell

Python 3.2.2 (default, Sep 4 2013 12:32)
Type "copyright", "credits" or "license()" for more information.
>>> 123+56
179
>>> 123-56
67
>>> 123*56
6888
>>> 123/56
2.1964285714285716
>>> 123/56
2
>>> |
```

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.
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 floating point number 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.

\[
\begin{align*}
\text{max} &= 1.7976931348623157e+308 \\
\text{min} &= 2.2250738585072014e-308
\end{align*}
\]

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.

```
>>> 123.4+56.7  
180.10000000000002  
>>> 123.4-56.7  
66.7  
>>> 123.4*56.7  
6996.780000000001  
>>> 123.4/56.7  
2.1763684303351  
>>> 123.4%56.7  
10.0  
>>> 123.4%1  
0.4000000000000057  
>>> 123.4%10  
3.400000000000057  
>>> 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:

**String:** A string is a sequence of one or more characters. An individual character is also a string 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:

<table>
<thead>
<tr>
<th>Python Shell</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;&gt;</td>
</tr>
<tr>
<td>&gt;&gt;&gt; 'hello'</td>
</tr>
<tr>
<td>'hello'</td>
</tr>
<tr>
<td>&gt;&gt;&gt; &quot;hello&quot;</td>
</tr>
<tr>
<td>'hello'</td>
</tr>
<tr>
<td>&gt;&gt;&gt; &quot;'hello'&quot;</td>
</tr>
<tr>
<td>'hello'</td>
</tr>
<tr>
<td>&gt;&gt;&gt; &quot;&quot;&quot;hello&quot;&quot;&quot;</td>
</tr>
<tr>
<td>'hello'</td>
</tr>
<tr>
<td>&gt;&gt;&gt; 'hello'</td>
</tr>
</tbody>
</table>

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 probably the most important one: “+” called **concatenation**.

What about 3*’Hello’? Can we “multiply strings?”

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

How come???????
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 comparison operations?

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

\( x \leq y < z \) 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 \( x \text{ and } y \) first evaluates \( x \); if \( x \) is false, its value is returned; otherwise, \( y \) is evaluated and the resulting value is returned.

The expression \( x \text{ or } y \) first evaluates \( x \); if \( x \) is true, its value is returned; otherwise, \( y \) is evaluated and the resulting value is returned.
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()
```

The function `type(x)` returns the “type”, i.e. the kind of object x is.
How do we get Python to output 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.**

```python
print('Hello world')
```

**Here is the syntax of the print statement.**

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

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 “passed in”. If there is more than one argument, they are separated from each other by a comma.

**What do the arguments mean/control?**
value

sep

end

file

flush

What is a default value:

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: