

6.189 Day 1

Readings

How To Think Like A Computer Scientist, chapters 1 and 2; chapter 4.12.

What to turn in

Turn in a printout of your code stapled to the last page of this handout (which should have your written answers to Exercises 1.6, 1.7, and 1.8).

Exercise 1.0 – Installing Python

Follow the instructions on installing Python and IDLE on your own computer on the Materials page of the course website. Be sure to install Python 2.6.4. Ask an LA for help if you run into any trouble.

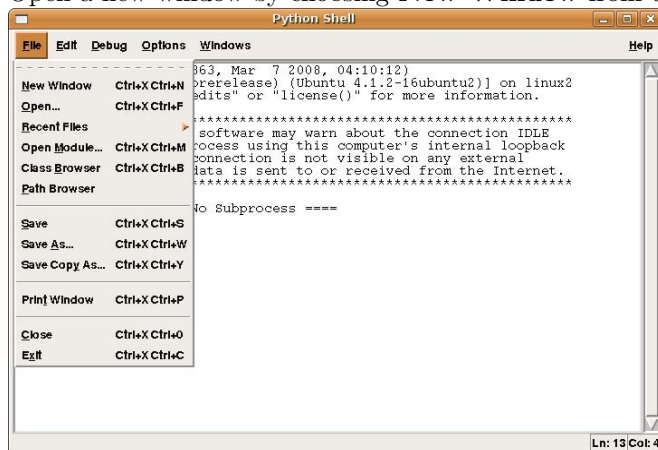
Exercise 1.1 – Hello, world!

Recall that a program is just a set of instructions for the computer to execute. Let's start with a basic command:

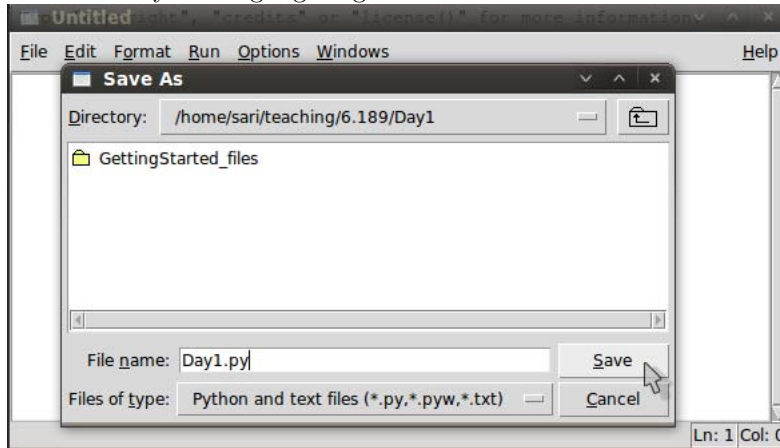
`print x`: Prints the value of the expression `x`, followed by a new line.

Create a new program called `Day1.py`. You will use this file to write your very first 'Hello, world!' program, as well as your answers for the rest of the exercises for today. How to create a program file:

1. Open a new window by choosing **New Window** from the **File** menu.



2. Save the file as `Day1.py`. Do NOT skip the `.py` portion of the file name - otherwise, you will lose out on syntax highlighting!



3. Start every program with a bank of comments, with a comment line for your name, the name of your file, and today's date. Recall that a comment line begins with a `#` (pound) symbol.

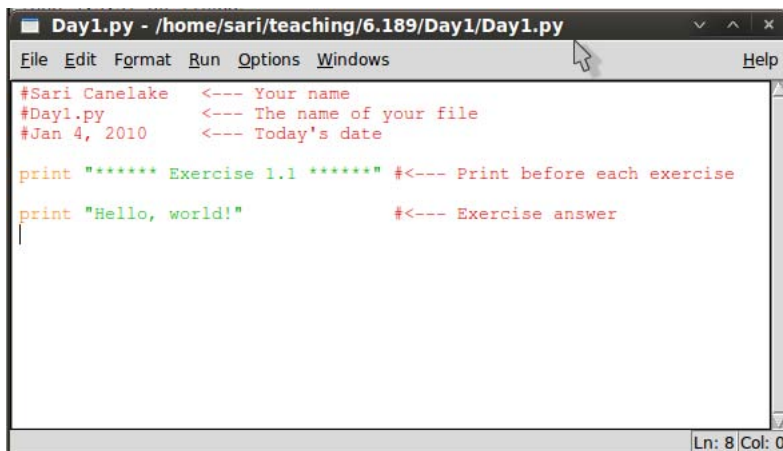
You may now write your very own Hello, world program! This is the first program that most programmers write in a new programming language. In Python, Hello world! is a very simple program to write!

Before you begin any exercise, please put the following line in your code:

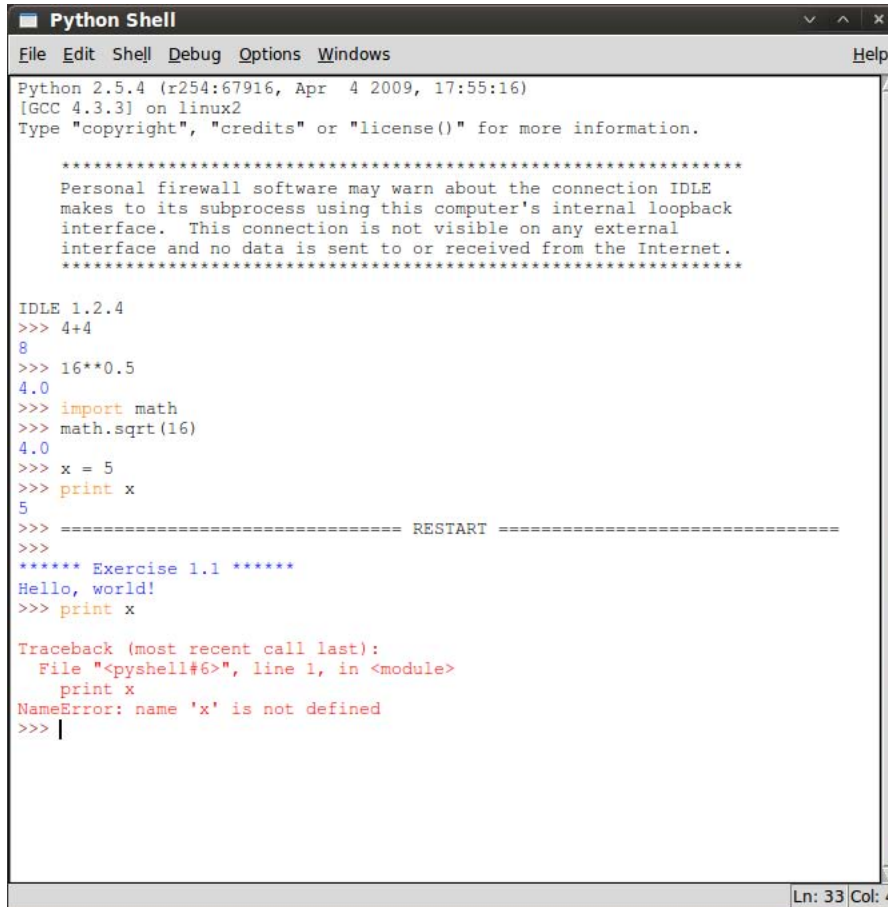
```
print "***** Exercise 1.1 *****"
```

(replacing 1.1 with whatever exercise you are working on). This makes it much easier for us to grade your work, and for you to keep your work organized! You may now write your Hello, world! program- it should be only one line!

When you are done, save your work and run it. To run your program, chose **Run Module** from the **Run** menu (or just hit F5 on Windows/Linux, or fn-F5 on a Mac). Your code should look similar to this:



and your shell should look similar to this:



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.5.4 (r254:67916, Apr 4 2009, 17:55:16)
[GCC 4.3.3] on linux2
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.2.4
>>> 4+4
8
>>> 16**0.5
4.0
>>> import math
>>> math.sqrt(16)
4.0
>>> x = 5
>>> print x
5
>>> ===== RESTART =====
>>>
***** Exercise 1.1 *****
Hello, world!
>>> print x

Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print x
NameError: name 'x' is not defined
>>> |
```

We'll go more into depth about variables soon, but note in this example how the variable *x* is undefined after we restart the shell.

Exercise 1.2 – Printing

New exercise! Be sure to add a new line to your code like this:

```
print "***** Exercise 1.2 *****"
```

Write a program that, when run, prints out a tic-tac-toe board. Remember to save your program regularly, to keep from losing your work! The purpose of this exercise is to make sure you understand how to write programs using your computing environment; many students in introductory courses experience trouble with assignments not because they have trouble with the material, but because of some weird environment quirk.

Expected output:

```
| |
-----
| |
-----
| |
```

Exercise 1.3 – Variables

Recall that variables are containers for storing information. For example,

Program Text:

```
a = "Hello, world!"
print a
```

Output:

```
Hello, world!
```

The = sign is an assignment operator which tells the interpreter to assign the **value** ‘‘Hello, world!’’ to the variable *a*.

Program Text:

```
a = "Hello, world!"
a = "and goodbye..."
print a
```

Output:

```
and goodbye...
```

Taking this second example, the value of *a* after executing the first line above is ‘‘Hello, world!’’. But, after executing the second line, the value of *a* changes to ‘‘and goodbye...’’. Since we ask the program to print out *a* only after the second assignment statement, that is the value that gets printed. If you wanted to save the values of both strings, you should change the second variable to another valid variable name, such as *b*.

Variables are useful because they can cut down on the amount of code you have to write. Write a program that prints out the tic-tac-toe board from exercise 1.2, but which uses variables to cut down on the amount of typing you have to do.

Exercise 1.4 – Operators/Order of Operation

Python has the ability to be used as a cheap, 5-dollar calculator. In particular, it supports basic mathematical operators +, -, *, / as well as the power operator (**) and the modulus operator (%).

Program Text:

```
x = 5 + 7
print x
y = x + 10
print y
```

Output:

```
12
22
```

Note that we can use variables in the definition of other variables! Mathematical operators only work on numbers- *ints* or *floats*. Statements such as 'Hi' + 5 or '5' + 7 will not work.

Part I: Transcribe the following equations into Python (without simplifying!), preserving order of operation with parenthesis as needed. Save each as the value of a variable, and then print the variable.

1. $\frac{3 \times 5}{2 + 3}$
2. $\sqrt{7 + 9} \times 2$
3. $(4 - 7)^3$
4. $\sqrt[4]{-19 + 100}$
5. $6 \bmod 4$ - If you aren't familiar with modular arithmetic, it is pretty straightforward- the modulus operator, in the expression $x \bmod y$, gives the remainder when x is divided by y . Try a couple modular expressions until you get the hang of it.

Part II: Use order of operation mathematics to create two equations that look the same (ie, have the same numbers) but evaluate to different values. Save each as the value of a variable, then print the variables.

Part III: Input the following sets of equations, and note the difference between *int* arithmetic and *float* arithmetic. You can do this just in your interpreter (you don't need to turn anything in for this part), but pay attention to the output!

1. $\frac{5}{2}$, $\frac{5}{2.0}$, and $\frac{5.0}{2}$ - note that as long as one argument is a float, all of your math will be floating point!
2. $7 * \left(\frac{1}{2}\right)$ and $7 * \left(\frac{1}{2.0}\right)$
3. $5 * *2$, $5.0 * *2$, and $5 * *2.0$
4. $\frac{1}{3.0}$ - note the final digit is rounded. Python does this for non-terminating decimal numbers, as computers cannot store infinite numbers! Take 6.004 to find out more about this...

Exercise 1.5 – User input

In this exercise, we will ask the user for his/her first and last name, and date of birth, and print them out formatted. Recall that you can get input from the user using the command `raw_input('text')`, as shown in lecture. Here is an example of what the program would do:

Output:

```
Enter your first name: Chuck
Enter your last name: Norris
Enter your date of birth:
Month? March
Day? 10
Year? 1940
Chuck Norris was born on March 10, 1940.
```

To print a string and a number in one line, you just need to separate the arguments with a comma (this works for any two types within a print statement). The comma adds a space between the two arguments. For example, the line:
`print 'October', 20, 1977`

will have the output

```
October 20 1977
```

Note that none of the commas are in this output! To do that you want something like this:

```
print 'October', str(20)+'',',', 1977.
```

The `+` sign concatenates two strings, but can *only* be used on two strings. Using it on a number and a string will cause an error (because it is ambiguous as to what you want the program to do! See Exercise 1.8 for more on ambiguous language).

We first convert the number 20 to a string using the `str` operation. Then, we can concatenate the *string* 20 to the *string* ', '. We'll cover this more in-depth on Day 4, when we get to strings, but you may want to play with these operations now to get everything to look its prettiest.

Name & Athena username:

Tear off this sheet and turn it in at the start of lecture tomorrow. Print out and staple your code file to it.

Exercise 1.6 – Variable Names

The Python interpreter has strict rules for variable names. which of the following are legal Python names? if the name is not legal, state the reason.

1. `and`
2. `_and`
3. `var`
4. `var1`
5. `1var`
6. `my-name`
7. `your_name`
8. `COLOR`

Exercise 1.7 – Types

It is important that we know the type of the values stored in a variable so that we can use the correct operators (as we have already seen!). Python automatically infers the type from the value you assign to the variable. Write down the type of the values stored in each of the variables below. Pay special attention to punctuation: values are not always the type they seem!

1. `a = False`
2. `b = 3.7`

3. `c = 'Alex'`

4. `d = 7`

5. `e = 'True'`

6. `f = 17`

7. `g = '17'`

8. `h = True`

9. `i = '3.14159'`

To verify your answers, you can use the interactive Python shell, but first try to do the exercise without help.

```
>>> a = False
>>> type(a)
<type 'bool'>
>>>
```

Exercise 1.8 – Natural Language Processing

Consider the following sentence:

Alice saw the boy on the hill with the telescope.

1. Draw a sketch of what's described in this sentence.

2. Draw a different sketch that could also be described by this sentence.

3. Write the sentence in two different ways, that clarifies the meaning of each of your sketches, next to your above sketches (hint: rewrite the sentence using extra words, commas, etc).
4. The ambiguity illustrated by this sentence is known as “prepositional phrase attachment.” Think about this as you continue to learn how to program, and consider how programming languages are designed to avoid the ambiguity illustrated by this example!

Exercise 1.9 – Zeller’s Algorithm

OPTIONAL!- Some problem sets will have an optional exercise at the end. Feel free to work on these problems if you have time at the end of the assignment, but you certainly don’t have to do them. However, you will get excellent practice in Python!

Zeller’s algorithm computes the day of the week on which a given date will fall (or fell). In this exercise, you will write a program to run Zeller’s algorithm on a specific date. You will need to create a new file for this program, `zellers.py`. The program should use the algorithm outlined below to compute the day of the week on which the user’s birthday fell in the year you were born and print the result to the screen.

Start with the program in Exercise 1.5, but ask for the month as a number between 1-12 where March is 1 and February is 12. If born in Jan or Feb, enter previous year (see the notes below). In the end, print out the name of the user and on what they of the week they were born.

Note: There are two functions to get user input. The first, `raw_input`, turns whatever the user inputs into a string automatically. The second, `input`, *preserves type*. So, if the user inputs an int, or a float, you will get an int or a float (rather than a string). Be careful though- you still want to use `raw_input` if you want a string back, or otherwise the user will have to put quotes around their answer. `input` is handy for computing this math!

Zeller’s algorithm is defined as follows:

Let A, B, C, D denote integer variables that have the following values:

- A = the month of the year, with March having the value 1, April the value 2, . . . , December the value 10, and January and February being counted as months 11 and 12 of the preceding year (in which case, subtract 1 from C)
- B = the day of the month (1, 2, 3, . . . , 30, 31)
- C = the year of the century (e.g. C = 89 for the year 1989)
- D = the century (e.g. D = 19 for the year 1989)

Note: if the month is January or February, then the preceding year is used for computation. This is because there was a period in history when March 1st, not January 1st, was the beginning of the year.

Let W, X, Y, Z, R also denote integer variables. Compute their values in the following order using integer arithmetic:

- W = $(13 * A - 1) / 5$
- X = $C / 4$
- Y = $D / 4$
- Z = $W + X + Y + B + C - 2 * D$
- R = the remainder when Z is divided by 7

The value of R is the day of the week, where 0 represents Sunday, 1 is Monday, . . . , 6 is Saturday. If the computed value of R is a negative number, add 7 to get a non negative number between 0 and 6 (you don't need to do this in the code). Print out R. You can check to be sure your code is working by looking at <http://www.timeanddate.com/calendar/>.

Run some test cases- try today's date, your birth date, and whatever else interests you!

Feel free to submit your zellers.py code if you wish, we'll take a look at it if you do.

MIT OpenCourseWare
<http://ocw.mit.edu>

6.189 A Gentle Introduction to Programming Using Python
January (IAP) 2010

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.