Python Programming For Beginners

UCSC Extension Silicon Valley

Donald Keidel, Ph.D.

Syllabus

Contact Information:

Instructor: Donald Keidel, Ph.D.

Email Address: dkeidel@ucsc.edu

About This Course:

This class is designed for beginners or novices. These are individuals that have no prior programming experience. But if you have prior programming experience, maybe in a different language or in python, you can still learn a great deal from taking this course.

Like all new subjects in ones life, it takes hours of practice and failures to become familiar with programming languages. Many new concepts will be presented that will require dedication and experimentation to master. Do not hesitate to use all resources available to you. These include the course materials, other individuals taking the course, the instructor, and numerous internet sources. You might find that discussing your issue with others, or reading online from others that have the same or similar issue, allows you to see the problem in a new and different light.

Some Resources

1. *Python Programming for the Absolute Beginner* by Michael Dawson. ISBN-13: 978-1435455009. This book is not required, but if you would like a physical book this is a good one.
2. http://www.python.org
3. http://www.python.org/doc - documentation
4. http://www.python.org/ downloads – can install your own version of Python at home. I suggest version 2.X (where X is 6 or 7) since this is what we will be using in course
5. http://pythonbooks.revolunet.com/ - The best free Python Resources
6. Various other resources:
    a. http://nbviewer.ipython.org/gist/rpmuller/5920182
    b. https://wiki.python.org/moin/Python2orPython3
    c. http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html
    d. https://www.fullstackpython.com/python-2-or-3.html

Grading:

There are a total of 5 homework assignments. Your grade will be based on the average of the 4 best homework assignments. Meaning that you can skip one assignment or the lowest grade of the 5 will be dropped.

Homework:	80%
Class Participation:	20%

To turn in homework via UCSC Extension Online. You can access the course site by visiting https://ucsc-extension.instructure.com/login/canvas. Assignments are due before the beginning of class time. You must upload your code as well as the output.

Late homework will not be accepted.

To class also bring a paper copy of the homework with output. **You will be grading your own assignment**. When grading, mark the error and correct the error fixing it so that it works. You will receive a good grade if your corrected code is good.

Remember when working on your homework assignments to use all available resources: email, call, or meet with fellow students in course, search the web, communicate with other students via the course website, and email the instructor. You may also join the mailing list tutor@python.org - https://mail.python.org/mailman/listinfo/tutor. This list is for individuals who want to ask questions regarding how to learn computer programming with the Python language and its standard library.

Style Guide for all your programs. This was adapted from the other instructors Style Guide so that we remain consistent in our teaching of programming styles:

- All labels (names of variables) should be descriptive.
- Labels for numbers and strings should be all lower case with an underscore to separate words (e.g. first_name).
- Labels for constants should be all capitalized, also with an underscore to separate words.
- Labels for functions should be UpperCamelCase (e.g. GetFirstName).
- Labels for functions should include verbs (e.g. GetLastName).
- Try to write classes, methods of classes, or functions to house all your code.
- Labels for classes should also be UpperCamelCase (class ClassName:).
- Labels for classes should be nouns (class Person:).

Lab 1 – Language

Hardware and Software
Python
Writing to stdout: print
Literals(Strings) and Numbers
Operators: + * =

Overview:

- Our job as a programmer is to write statements in the Python language that will control our computer system.

- This lab describes the basic topics of what a computer is and how we set up a computer to perform a task.

Hardware Terminology

Computer, Computer System

- whole *system* of interconnected parts that make up a computer

- Without software, it's just a lump of parts

- *Processor* - controls most of what a computer does

- Other components include the *memory*

Memory, RAM

- The computer's working memory (*Random-Access Memory*, or RAM) contains two things:
    - our data
    - the processing instructions (or program) for manipulating that data
- Since the instructions are stored in memory, they can be changed.
    - Every time we double click an icon and a program is loaded into memory.
    - When we open a document file, we see it read from the disk into memory so we can work on it.
- Memory is *dynamic*: it changes as software runs.
- Memory which doesn't change is called *Read-Only Memory* (ROM).
- Memory is *volatile*: when we turn the computer off, the contents vanish.
- Memory is accessed "randomly": any of the millions of bytes of my computer's memory can be accessed with equal ease.

Software Terminology

- You can see and touch the hardware.
- Software, on the other hand, is less tangible.
- **Programming** is the act of creating new software.

Operating System

- The Operating System (OS) ties all of the computer's devices together
- The operating system:
    - Contains the software called *device drivers* that make the various devices work consistently.
    - *Manages resources* like memory and time by assuring that all the programs share those resources.
    - Manages the various disk drives by imposing some organizing rules on the data (*file system*)

- *Manages the various windows*; it directs mouse clicks and keyboard characters to the proper application program.
- Starts programs.
  - Starting a program means allocating memory
  - loading the instructions from the disk
  - allocating processor time to the program
  - and allocating any other resources in the processor chip.

Program, Application, Software

- Other names for "program":
  - "command"
  - "application"
  - "application program"
  - "application system"
  - "solutions"
- A program is rarely a single thing or file
  - We will think of a program as the one file that contains the *main* part of the program.

There are basically two types of programs:

1. *binary executable*
2. *script*

- A binary executable or *binary application* is a program that takes direct control of the computer's processor.
  - it uses the binary codes specific to the processor chip inside the computer.

The Python Program and What It Does

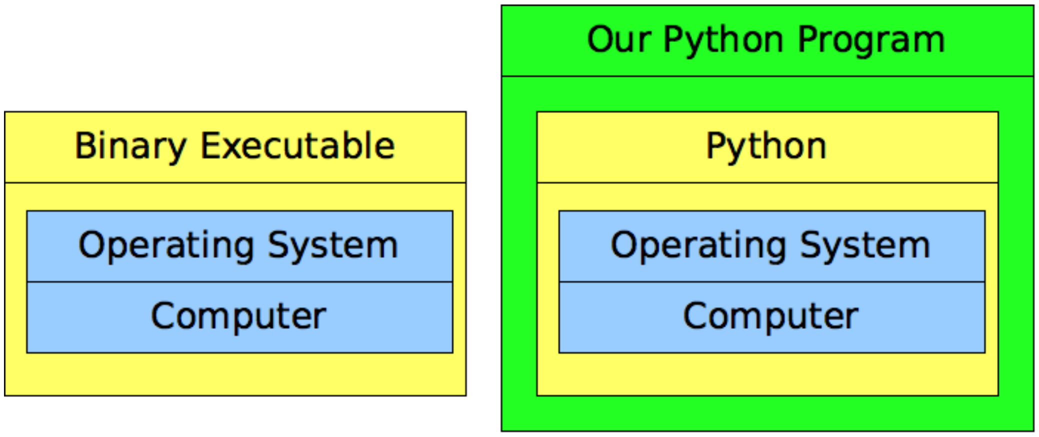
- Earlier we talked about binary executables and scripts.

    - binary program - uses codes that are specific to our processor chip and operating system.
    - Script - written in an easy-to-read language like the Python language.

- They work together since the scripts depends on a binary executable (i.e. Python).

- The **Python** program, python or python.exe, is an *interpreter* or a *virtual machine*.

    - The **Python** program's job is to read statements in the Python language and execute those statements.
    - This process is called "interpreting" the program



The Python Virtual Machine

When we write Python-language statements, those statements will control the **Python** program -> the **Python** program controls the OS kernel -> the OS kernel controls our computer.

- The *cost* of using a Python virtual machine is programs that are somewhat slower than those which use the computer's internal codes (binary executable).

- The *benefit* is a huge simplification in how we write and use software:

    - we don't have to understand the ins and outs of our processor chip
    - we can describe our data and processing clearly and succinctly.

Why Program in Python:

1. Python is an easy to learn programming language.

2. write complex operations (data structures) in fewer statements than in C, C++ or Java.

3. Object-oriented programming is a lot easier than in languages like Java.

4. clean syntax and code readability.

5. Python is a general-purpose high-level programming language.

6. Python is both object oriented and it can be even used in a functional style as well.

7. Python programs are portable (i.e. they can be ported to other operating systems like Windows, Linux, Unix and Mac OS X, and they can be run on Java and .NET virtual machines)

8. Python is very fast.
    a. The source code is compiled into bytecode, so that executing the same file will be faster, if the script will be executed again.
        i. The bytecode is an "intermediate language", which is said to run on a virtual machine that executes the machine code corresponding to each bytecode.

9. The Python Standard Library contains an enormous number of useful modules and is part of every standard Python installation

Integrated Development Environment (IDE)

An integrated development environment (IDE) or interactive development environment:

- software application that provides comprehensive facilities to computer programmers for software development
- consists of a source code editor, build automation tools and a debugger
- most IDEs offer Intelligent code completion features.

IDLE is an integrated development environment for Python

- bundled with the default implementation of the language since version 1.5.2b1.

- completely written in Python and the Tkinter GUI toolkit.

- IDLE is a simple IDE and developed for beginners in an educational environment.

- it is cross-platform.

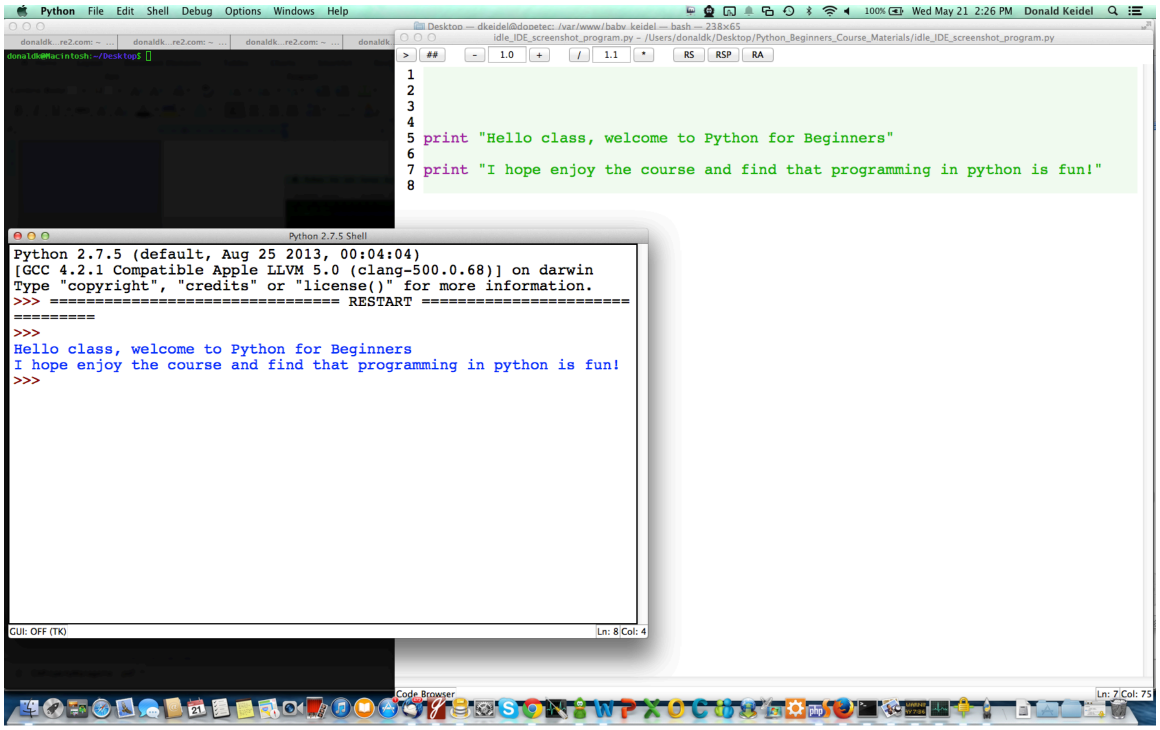
- its main features are:

    - Multi-window text editor with syntax highlighting
    - Autocompletion
    - smart indent
    - Python shell with syntax highlighting.
    - Integrated debugger with stepping
    - persistent breakpoints
    - call stack visibility.

Author Guido van Rossum says IDLE stands for "Integrated DeveLopment Environment", and since van Rossum named the language Python partly to honor British comedy group Monty Python, the name IDLE was probably also chosen partly to honor Eric Idle, one of Monty Python's founding members.

Visit this link to learn about other IDEs suitable for developing in Python:

https://en.wikipedia.org/wiki/Comparison_of_integrated_development_environme nts#Python



```
print "Hello class, welcome to Python for Beginners"

print "I hope enjoy the course and find that programming in python is fun!"
```

```
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ========================
=========
>>>
Hello class, welcome to Python for Beginners
I hope enjoy the course and find that programming in python is fun!
>>>
```

IDLE: Screenshot taken of IDLE running on Mac OS X Mavericks with Python 2.7

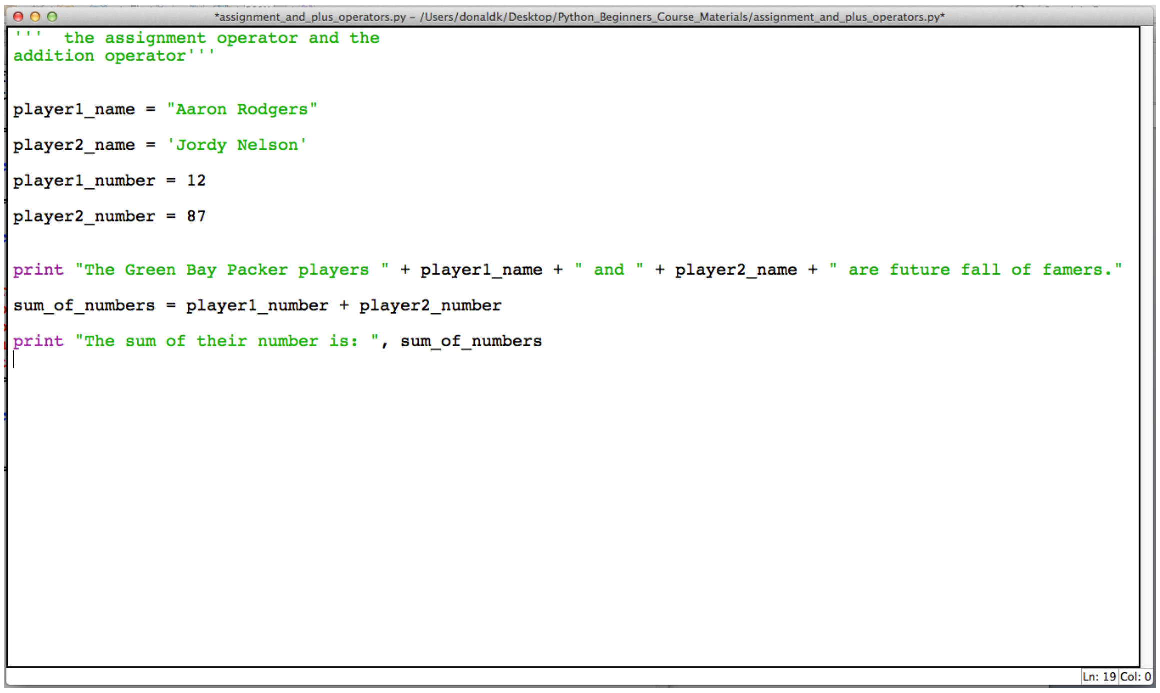
Donald Keidel, Ph.D. 2016

print

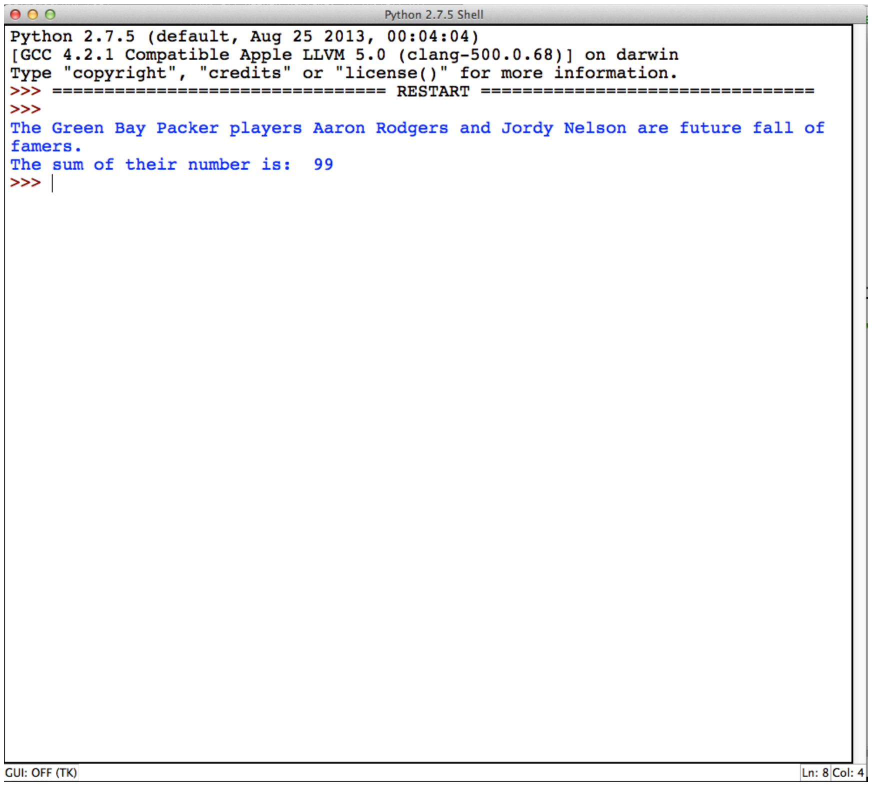
- Most computer programs communicate with the outside world.
- a program has to deliver its result in some way.
- One form of output goes to the *standard output* by using the print statement in Python.
- Starting with version 3.0, Python doesn't provide a print statement anymore, there is only a print function.

```
>>> print "Hello User"
Hello User
>>> answer = 42
>>> print "The answer is: " + str(answer)
The answer is: 42
>>>
```

assignment operator (=) and plus operator (+)


assignment_and_plus_operators.py - /Users/donaldk/Desktop/Python_Beginners_Course_Materials/assignment_and_plus_operators.py
Ln: 19 Col: 0

```
'''  the assignment operator and the
addition operator'''


player1_name = "Aaron Rodgers"

player2_name = 'Jordy Nelson'

player1_number = 12

player2_number = 87


print "The Green Bay Packer players " + player1_name + " and " + player2_name + " are future fall of famers."

sum_of_numbers = player1_number + player2_number

print "The sum of their number is: ", sum_of_numbers
```



```
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
The Green Bay Packer players Aaron Rodgers and Jordy Nelson are future fall of
famers.
The sum of their number is:  99
>>>
```

Lab 1 – Exercises

1. What if you wanted to print a string, and this string contained quotation marks. How do you think you would do that?
2. Write a program that will print out the following:

   “Is there a way in which I can print out strings in python
   that contain more than one line
   without having to
       use 4 print statements?”

3. I find myself many times using the python interpreter as a calculator to perform simple math.

   Use the interpreter to sum up 52, 12, and 87.

   Then write a small script in IDLE that will do the same thing.

4. In addition to the = and + operators, we need to know how to use the – (subtraction), * (multiplication), and / (division) operators. What do you think you will get if you do the following:

   1. Multiplication of two integers

      num1 = 2
      num2 = 3

      num1 * num2 = ?

   2. Now lets involve a string

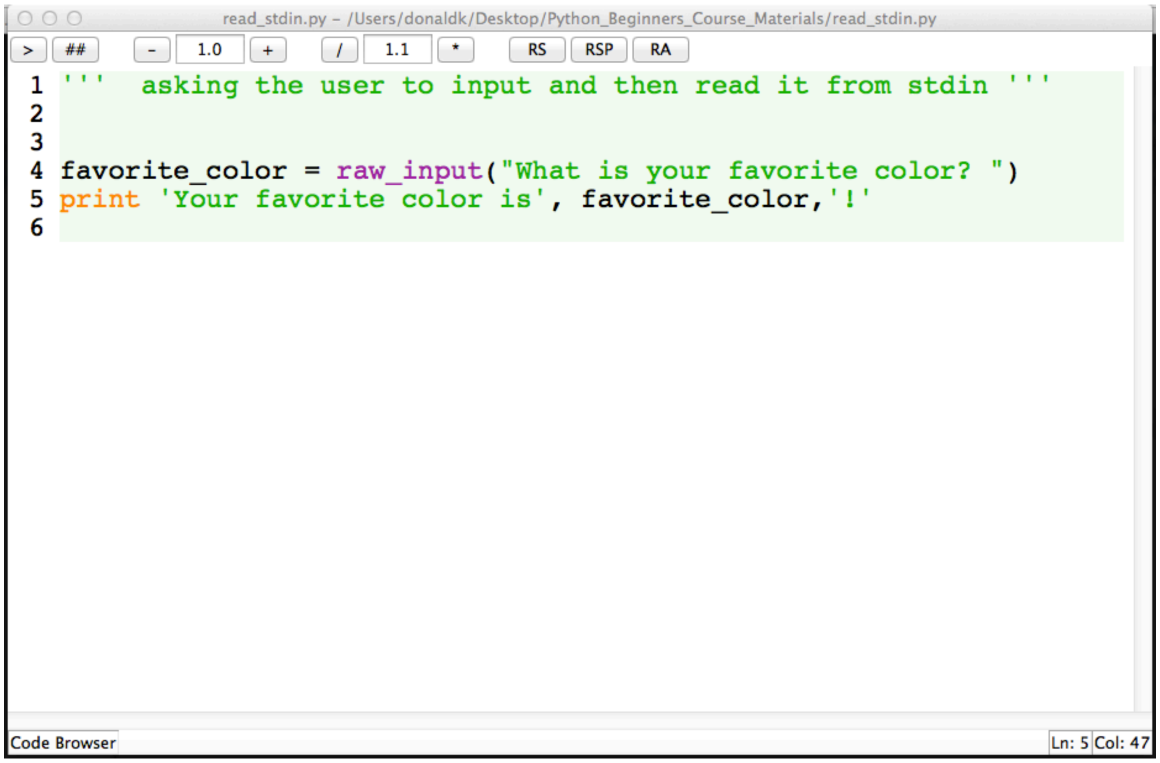
      string1 = ‘hello ‘

      string1 * num2 = ?

Lab 2 – Input

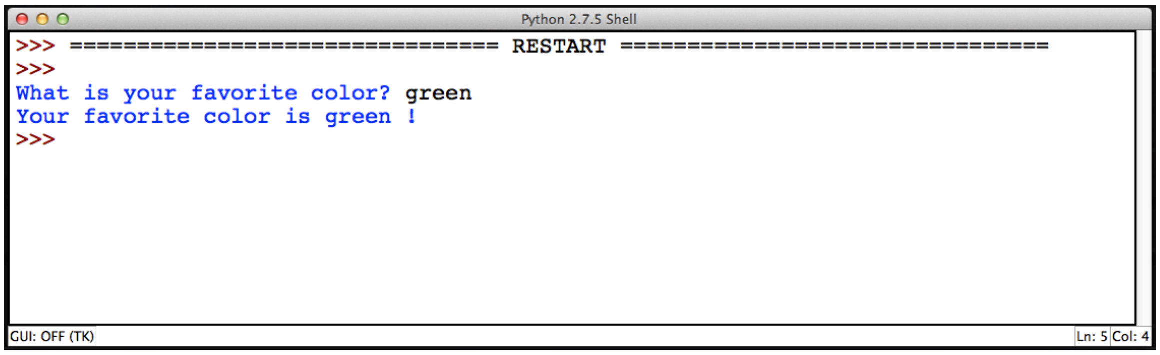
Reading from stdin
Reading a string
Reading a number
Handling errors
Modulo operator

Reading from standard in:



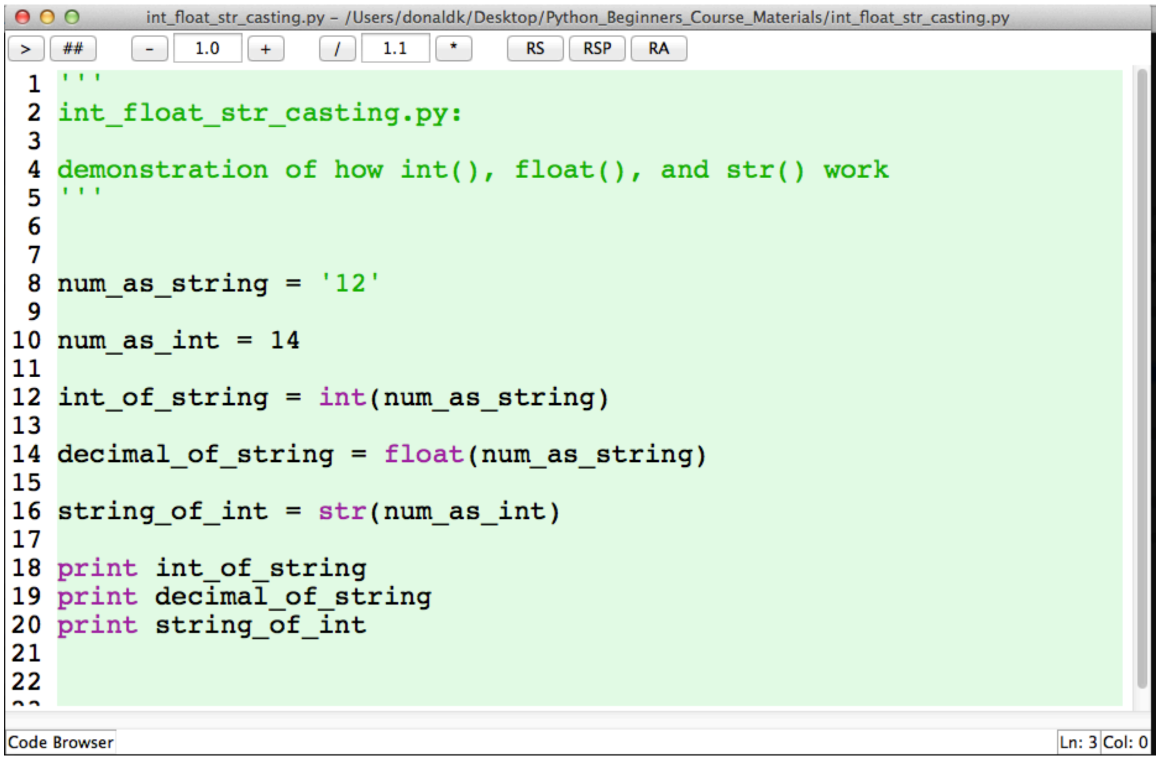
read_stdin.py – /Users/donaldk/Desktop/Python_Beginners_Course_Materials/read_stdin.py
Code Browser
Ln: 5 Col: 47

```python
'''  asking the user to input and then read it from stdin '''


favorite_color = raw_input("What is your favorite color? ")
print 'Your favorite color is', favorite_color,'!'

```



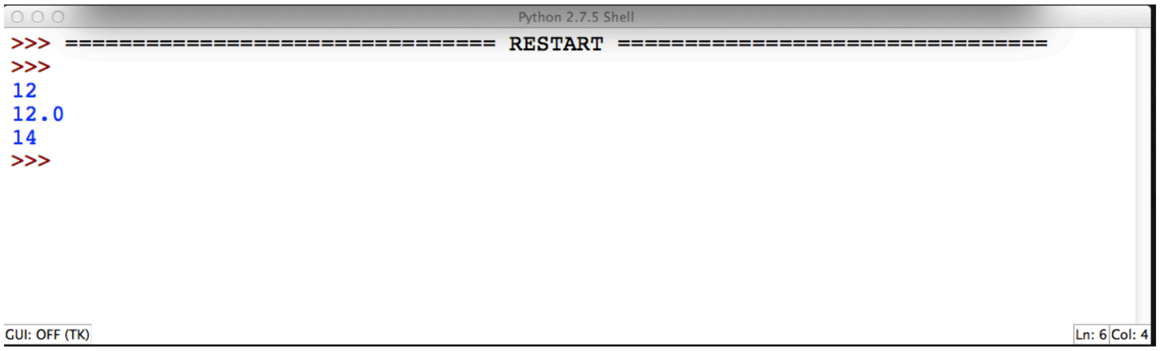
```
>>> ================================ RESTART ================================
>>> 
What is your favorite color? green
Your favorite color is green !
>>> 
```

type conversions:



int_float_str_casting.py - /Users/donaldk/Desktop/Python_Beginners_Course_Materials/int_float_str_casting.py

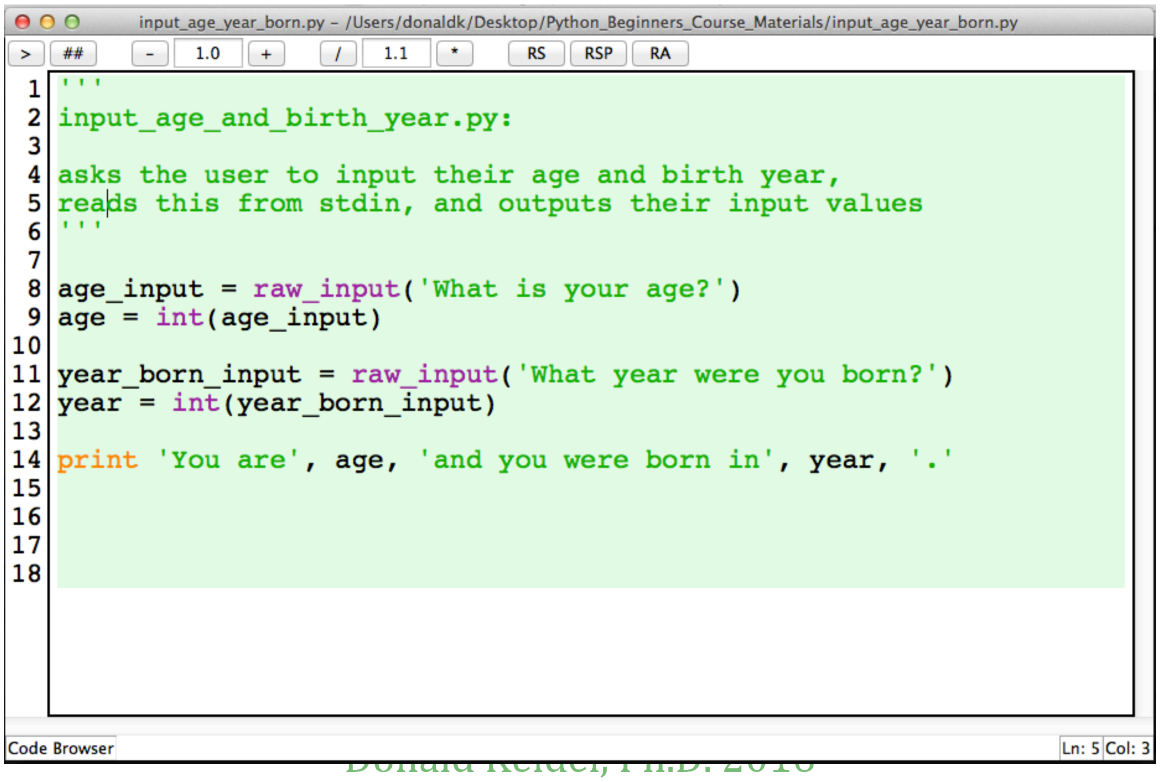
```python
'''
int_float_str_casting.py:

demonstration of how int(), float(), and str() work
'''


num_as_string = '12'

num_as_int = 14

int_of_string = int(num_as_string)

decimal_of_string = float(num_as_string)

string_of_int = str(num_as_int)

print int_of_string
print decimal_of_string
print string_of_int
```



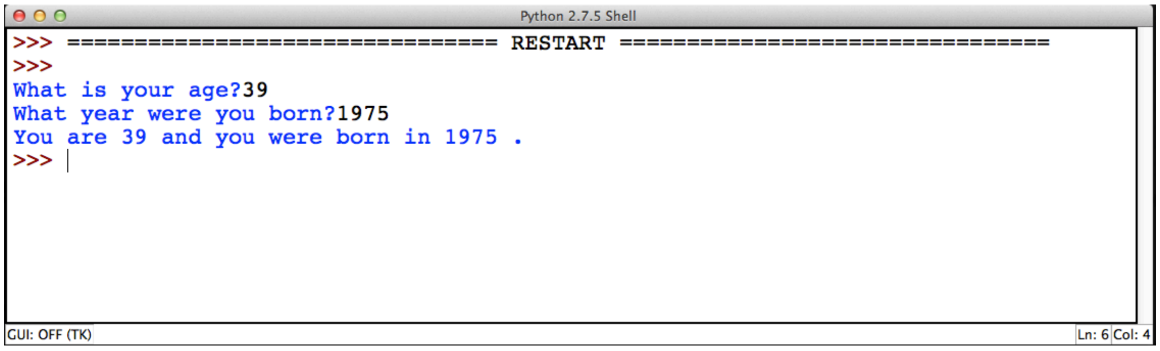
Python 2.7.5 Shell

```
>>> ================================ RESTART ================================
>>>
12
12.0
14
>>>
```

reading numbers from stdin:


input_age_year_born.py - /Users/donaldk/Desktop/Python_Beginners_Course_Materials/input_age_year_born.py
Code Browser
Ln: 5 Col: 3

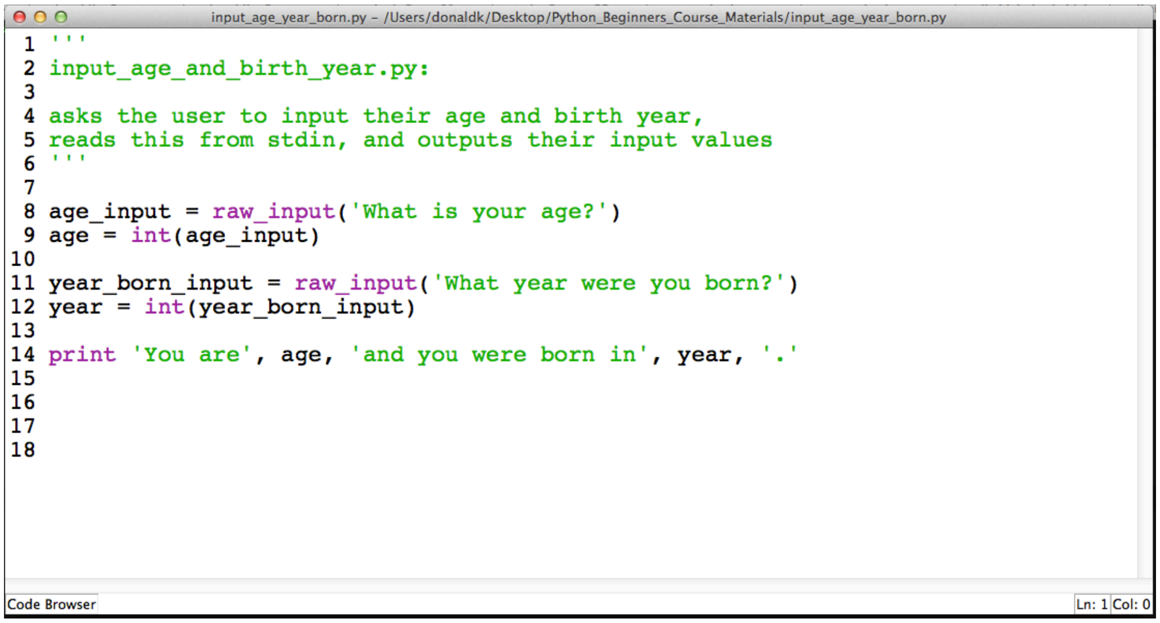
```python
'''
input_age_and_birth_year.py:

asks the user to input their age and birth year,
reads this from stdin, and outputs their input values
'''

age_input = raw_input('What is your age?')
age = int(age_input)

year_born_input = raw_input('What year were you born?')
year = int(year_born_input)

print 'You are', age, 'and you were born in', year, '.'
```



```
>>> ================================ RESTART ================================
>>> 
What is your age?39
What year were you born?1975
You are 39 and you were born in 1975 .
>>> 
```

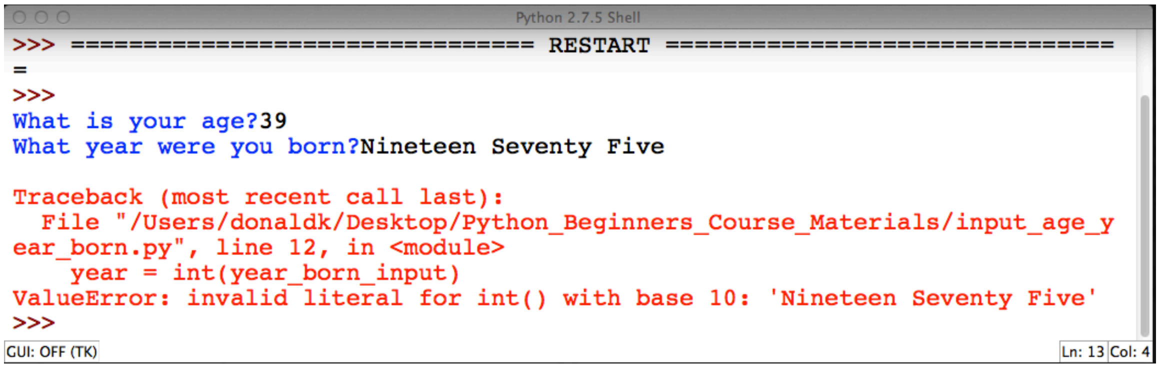
What do you think will happen if the user inputs something other than an integer for their age or the year they were born? After all, they are free to type whatever they like.

Reading strings as numbers from stdin:



input_age_year_born.py - /Users/donaldk/Desktop/Python_Beginners_Course_Materials/input_age_year_born.py

```python
'''
input_age_and_birth_year.py:

asks the user to input their age and birth year,
reads this from stdin, and outputs their input values
'''

age_input = raw_input('What is your age?')
age = int(age_input)

year_born_input = raw_input('What year were you born?')
year = int(year_born_input)

print 'You are', age, 'and you were born in', year, '.'
```



Python 2.7.5 Shell

```
>>> ================================ RESTART ================================
=
>>>
What is your age?39
What year were you born?Nineteen Seventy Five

Traceback (most recent call last):
  File "/Users/donaldk/Desktop/Python_Beginners_Course_Materials/input_age_y
ear_born.py", line 12, in <module>
    year = int(year_born_input)
ValueError: invalid literal for int() with base 10: 'Nineteen Seventy Five'
>>>
```

The Python built in function int() cannot convert 'Nineteen Seventy Five' string to an integer since it is not a whole number or a number that is not a fraction (integer).

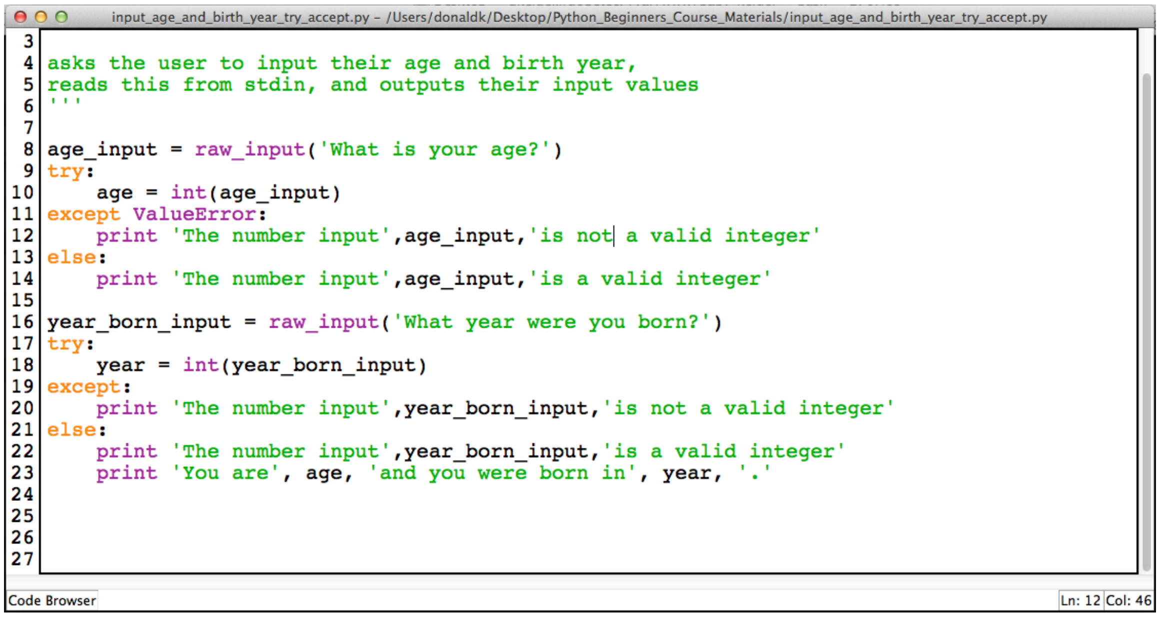
Errors detected during execution are called *exceptions* and are not unconditionally fatal. A very nice feature of Python is how it has exceptions built into it at the lowest levels. These are thrown and the program stops. But you can take advantage of this and catch the exception, which will allow for catching errors, while allowing the program to continue.

In Python we catch these exceptions using try, except, else. Here is how it is done:

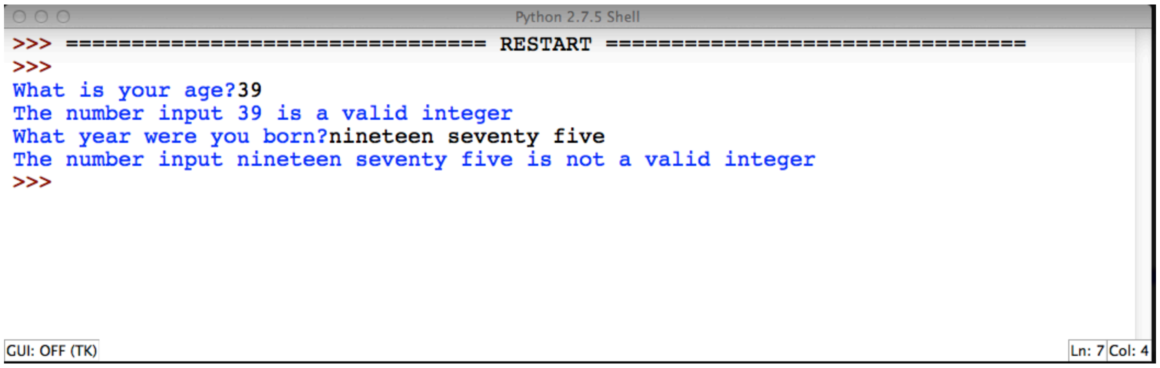
```
try:
  You do your operations here;
  ......................
except ExceptionI:
  If there is ExceptionI, then execute this block.
except ExceptionII:
  If there is ExceptionII, then execute this block.
  ......................
else:
  If there is no exception then execute this block.
```

Here are few important points about the above-mentioned syntax:

- A *single* try statement can have *multiple* except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- You can also provide a *generic* except clause, which handles any exception.
- After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block *does not* raise an exception.
- The else-block is a good place for code that *does not* need the try: block's protection.


input_age_and_birth_year_try_accept.py - /Users/donaldk/Desktop/Python_Beginners_Course_Materials/input_age_and_birth_year_try_accept.py

```python

asks the user to input their age and birth year,
reads this from stdin, and outputs their input values
'''

age_input = raw_input('What is your age?')
try:
    age = int(age_input)
except ValueError:
    print 'The number input',age_input,'is not a valid integer'
else:
    print 'The number input',age_input,'is a valid integer'

year_born_input = raw_input('What year were you born?')
try:
    year = int(year_born_input)
except:
    print 'The number input',year_born_input,'is not a valid integer'
else:
    print 'The number input',year_born_input,'is a valid integer'
    print 'You are', age, 'and you were born in', year, '.'
```


Python 2.7.5 Shell

```
>>> ================================ RESTART ================================
>>> 
What is your age?39
The number input 39 is a valid integer
What year were you born?nineteen seventy five
The number input nineteen seventy five is not a valid integer
>>> 
```

Modulus:

The Modulus operator (sometimes called Modulo) defines an operation which returns the **remainder** of a division of one number by another.

For example if we take the dividend 'a', and a divisor 'n' then we can define a % n (a *modulo* n) as the remainder.

So if we have 7 % 3 then we look at it as an integer division of only the proportion of the numbers which can evaluate to a whole number. In this case we see that 7 / 3 doesn't equal a whole number (equals 2.333) however 6/3 does (leaving no remainder) so to find the modulus of this expression we can treat 7 / 3 as 6/3 + remainder which tells us that the remainder is 1. Here's a few more examples just to help you get your head around the idea:
8 % 4 = 0
12 % 16 = 12
So a few rules to bear in mind which will help you use the operator in practice:
1. if a / n equals a whole number then a % n always equals 0
2. if a is less than n then a % n always equals a
3. if a is greater than n then a % n always equals a value between 1 and n-1

A more complex example is as follows, but you will not be required to calculate this. I included it in case you were curious:

What is −9 mod 6 ?

What remainder do we get when we divide −9 by 6 ?
When we divide a dividend, A, by a divisor, B, we get a quotient, Q, with a remainder, R.
$A/B=Q$ with a remainder of R where $0\leq R<B$
If we can find the remainder, R, in the expressions above then we can calculate A mod B since:
$A \text{ mod } B=R$
Our dividend, $A=-9$
Our divisor, $B=6$
If we can calculate Q, then we can calculate R
We can rewrite our expression as: $A=Q\cdot B+R$
or alternatively as: $A/B=Q+R/B$
This shows us how to find Q. Since $0\leq R<B$ we have: $0\leq R/B<1$
If we use use decimal division to divide A by B, then Q will be:
$Q=\lfloor A/B\rfloor$
$\lfloor x\rfloor$ means the next smallest integer less than or equal to x:
e.g. $\lfloor 2\rfloor=2$, $\lfloor 7.25\rfloor=7$, $\lfloor -5.1\rfloor=-6$
$Q=\lfloor -9/6\rfloor$
$Q=-2$
$A-9=Q\cdot B+R=-2\cdot 6+R$

$-9=-2\cdot 6+3$
$-9 \bmod 6=3$

Lab 2 – Exercises

1. Find the modulo for the following:

0 % 5 =	3 % 5 =	6 % 5 =
1 % 5 =	4 % 5 =	7 % 5 =
2 % 5 =	5 % 5 =	8 % 5 =

2. Write a program that asks the user for an integer. Determine if the integer is odd or even.

3. Write a small program that asks the user for their height. First ask the user for the height portion in feet, and then ask for the inches portion. Convert this height to meters. There are 39.3701 inches in 1 meter.

4. Write a program that asks the user for two integers. Then apply all the operators we have learned or heard about thus far: addition, subtraction, multiplication, division, and modulo. Make sure to error check when performing modulo. You need to consider a situation where the second number (n in notes) is zero.

5. Write a program or use the interpreter to determine the result of:

   4/2 =

   and

   5/2 =

   Are these answers accurate? How could you make them more accurate?

Homework 1:

Write a program that asks the user for their dream job title. Then ask them for the hourly wage they want to earn. This number should account for dollars and cents. Calculate the annual wage they will earn over a year. Assume they work 48 weeks a year (4 weeks vacation) but are paid over 52 weeks. Then ask them how much money they feel they will need at retirement time. Determine the number of years they will have to work to save up their retirement dollar value assuming their only source of income was their paycheck? Finally, determine if the number of years they need to save is an even or odd number. For this part, use the modulus operator to determine if the number you calculated is odd or even. Once you have calculated it, use print to print out the value. You should also print out something like this. "If the value is --- then the number is odd, however, if the value is ----- then the number is even."

Make sure you account for user input error. Test your code by trying to cause errors in input. It is not important to error check on their dream job title. Just assume it will be a reasonable string.

Print out a copy of your homework and bring it to class. Make sure to include the output.