

Introduction to Python 3

Week 3

Questions about last 2 weeks

Type Conversion – Most Common Ones

• Function	Description
• <code>int(x [,base])</code>	Converts x to an integer base specifies the base if x is a string.
• <code>long(x [,base])</code>	Converts x to a long integer base specifies the base if x is a string.
• <code>float(x)</code>	Converts x to a floating-point number.
• <code>hex(x)</code>	Converts an integer to a hexadecimal string.
• <code>str(x)</code>	Converts object x to a string representation.
• <code>chr(x)</code>	Converts an integer to a character.
• <code>complex(real [,imag])</code>	Creates a complex number.

Type Conversion – Less Common Ones

• Function	Description
• repr(x)	Converts object x to an expression string.
• eval(str)	Evaluates a string and returns an object.
• tuple(s)	Converts s to a tuple.
• list(s)	Converts s to a list.
• set(s)	Converts s to a set.
• dict(d)	Creates a dictionary d must be a sequence of (key,value) tuples.
• frozenset(s)	Converts s to a frozen set.
• unichr(x)	Converts an integer to a Unicode character.
• ord(x)	Converts a single character to its integer value.
• oct(x)	Converts an integer to an octal string.

Size of numbers

- Unlike other programming languages
- Python has no real limit on the size of numbers
- In extreme cases, you may run out of memory
- However an there are limits on how number can be displayed or converted

Size of numbers example

```
size_of_numbers.py - /Volumes/raid_blk/data/python_class/code_examples/size_of...  
big_integer= 10000000000000000000000000000000000000000000000000000000  
print("big integer is ",big_integer)  
print(" ")  
big_integer_plus_one = big_integer+1  
print("big integer +1 is ", big_integer_plus_one)  
print(" ")  
big_float=10**3000  
print("big float is ",big_float)  
#print("big float in exponential notation is ", "%10.0E"% big_float)
```

Ln: 5 Col: 0

Output of Size of numbers Example

[illegible]

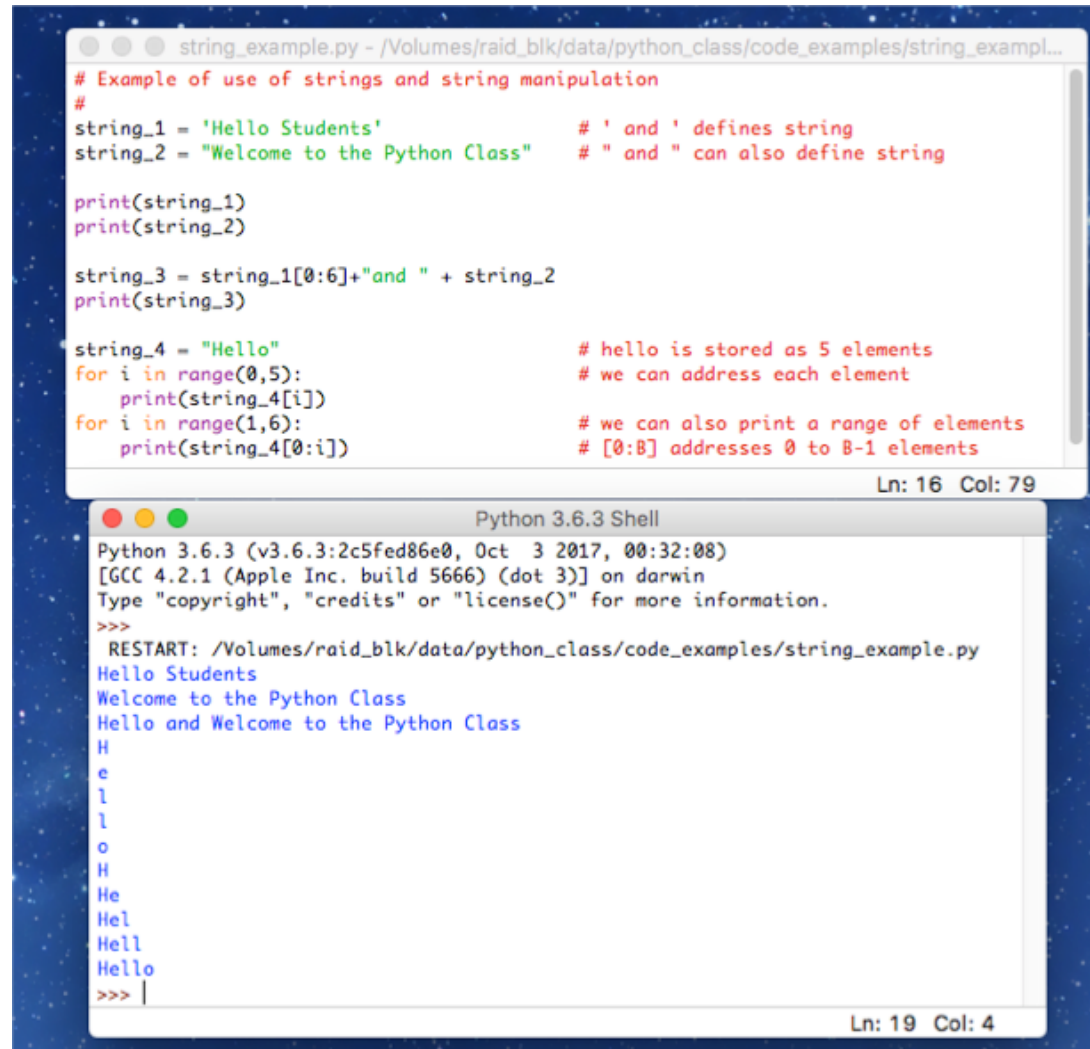
Strings and String Manipulation

- Strings are needed to give context to printed output
- Strings are also useful to manipulate data: name, address, etc.
- Strings are stored as a set of individual elements
- Therefore, each element in the string can be addressed individually
- For Example:
 - `string_4 = Hello`
- Then the value of:
 - `string_4[0]` is H,
 - `string_4[1]` is e,
 - `string_4[2]` is l,
 - `string_4[3]` is l,
 - `string_4[4]` is 0,
 - `String_4[0:4]` is Hel

Strings and String Manipulation (Con't)

- We can append to (add to) strings
 - `string_1 = "Hello"`
 - `string_2 = "Hel"`
 - `string_3 = string_1 + string_2` # works
- But can not directly subtract them
 - `string_3 = string_1 - string_2` # does not work
 - `string_3 = string_1[3:5]` # works

Example of String Manipulation



The image shows a screenshot of a Python script and its execution. The top window is a text editor showing the script `string_example.py`. The script demonstrates string manipulation, including defining strings, printing them, concatenating them, and iterating over characters and substrings. The bottom window is a Python 3.6.3 Shell showing the output of the script.

```
string_example.py - /Volumes/raid_blk/data/python_class/code_examples/string_exempl...  
# Example of use of strings and string manipulation  
#  
string_1 = 'Hello Students'          # ' and ' defines string  
string_2 = "Welcome to the Python Class"  # " and " can also define string  
  
print(string_1)  
print(string_2)  
  
string_3 = string_1[0:6]+"and " + string_2  
print(string_3)  
  
string_4 = "Hello"                   # hello is stored as 5 elements  
for i in range(0,5):                 # we can address each element  
    print(string_4[i])  
for i in range(1,6):                 # we can also print a range of elements  
    print(string_4[0:i])              # [0:8] addresses 0 to 8-1 elements  
  
Ln: 16 Col: 79
```

```
Python 3.6.3 Shell  
Python 3.6.3 (v3.6.3:2c5fed86e0, Oct 3 2017, 00:32:08)  
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin  
Type "copyright", "credits" or "license()" for more information.  
>>>  
RESTART: /Volumes/raid_blk/data/python_class/code_examples/string_example.py  
Hello Students  
Welcome to the Python Class  
Hello and Welcome to the Python Class  
H  
e  
l  
l  
o  
H  
He  
Hel  
Hell  
Hello  
>>> |  
  
Ln: 19 Col: 4
```

Lists, Tuples, and Dictionaries

- Lists are a list of values
 - Each one of them is numbered, starting from zero - the first one is numbered zero, the second 1, the third 2, etc
 - You can remove values from the list, and add new values to the end
- Tuples are similar to lists: you can't change their values, but you can create new ones
 - The values that you give it first remain for the rest of the program
 - Numbering is the same as lists, 0, 1, 2 ...
- Dictionaries are similar to what their name suggests - a dictionary
 - In a dictionary, you have an 'index' of words, and a corresponding definition
 - The word is called a 'key', and the definition a 'value', thus a (key, value) pair
 - The values in a dictionary aren't numbered or in any specific order
 - The key is used to reference the value
 - Values can added, removed, and modified

Lists

- Lists are modifiable, so their values can be changed.
- Most of the time we use lists, not tuples, because we want to easily change the values of things
- For example:
- `pet_names = ['Ralph', 'Tom', 'Blackie', 'Boots']`
- We address pets by `pet_names[0]`, `pet_names[1]`, etc.
- We can add or subtract from the list
- `pet_names.append('Bonnie')` adds to the end of the list
- `del pet_names[2]` deletes Blackie

Tuples

- Tuples are used for information that does not change
- Examples:
 - Zip codes
 - Days of the week
 - Months of the year
 - Past Presidents
- `days = ('Sunday' , 'Monday' , 'Tuesday' , 'Wednesday' , 'Thursday' , 'Friday' , 'Saturday')`

Dictionaries

- Dictionaries are addressed by their key
- For each Dictionary entry there is a key and a value
- For example
- `pets = {'Ralph':'Dog', 'Tom':'Cat', 'Blackie':'Dog', 'Boots':'Cat'}`
- We can add and subtract entries to the Dictionary
- Since we are addressing the Dictionary by the key, there is no numerical order to the entries
- `pets['Charlie'] = 'Dog'` adds new dog named Charlie
- `del pets['Tom']` deletes the cat named Tom

Example Code for Lists, Tuples, and Dictionaries

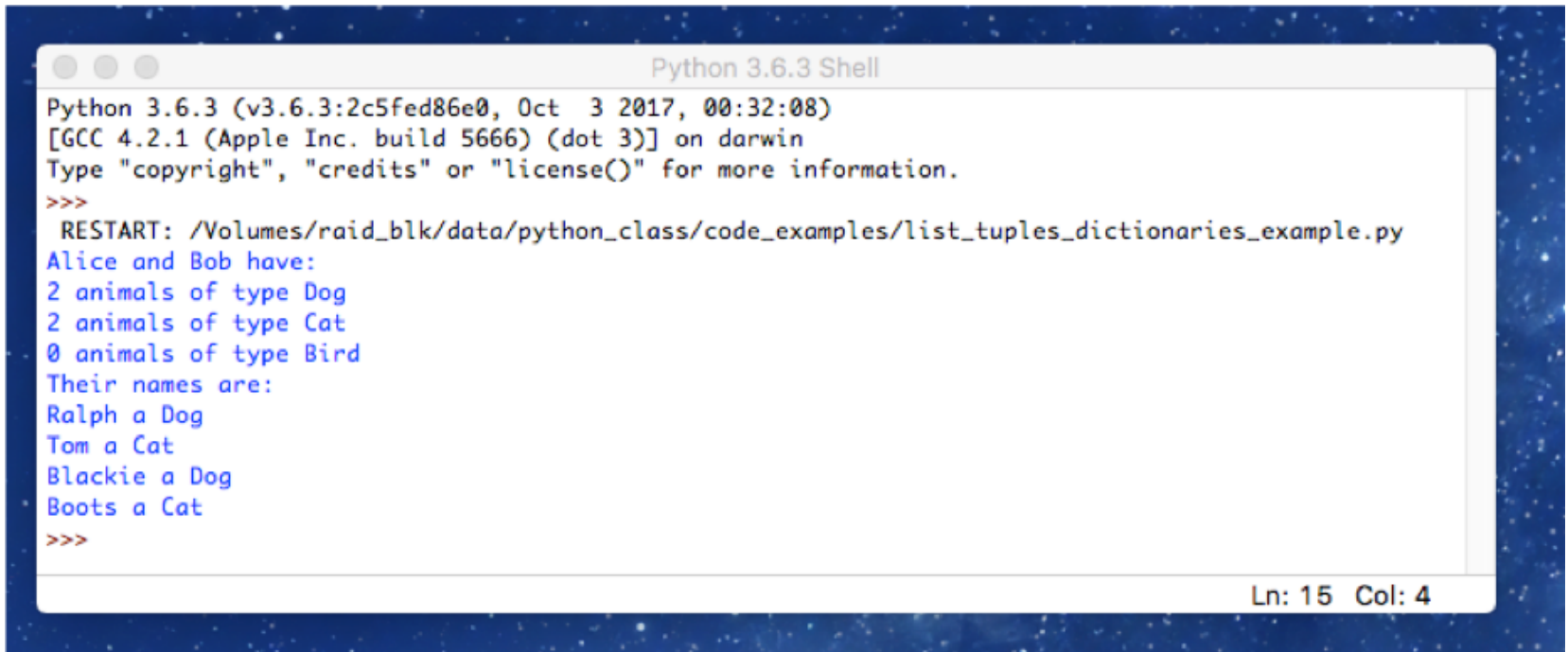
```
list_tuples_dictionaries_example.py - /Volumes/raid_blk/data/python_class/code_examples/list_tuples_dictionaries_example.py (3.6.3)
#Example Code for Lists, Tuples, and Dictionaries
pet_owners = ['Alice', 'Bob']
pet_types = ['Dog', 'Cat', 'Bird']
pet_names = ['Ralph', 'Tom', 'Blackie', 'Boots']
pets = {pet_names[0]:pet_types[0], pet_names[1]:pet_types[1], pet_names[2]:pet_types[0], pet_names[3]:pet_types[1]}
#
number_of_pets = len(pet_names)
#
number_of_dogs = 0
number_of_cats = 0
number_of_birds = 0

for pet_count in range (0, number_of_pets) :
    if (pets[pet_names[pet_count]] == pet_types[0]):
        number_of_dogs = number_of_dogs + 1
    elif (pets[pet_names[pet_count]] == pet_types[1]):
        number_of_cats = number_of_cats + 1
    elif (pets[pet_names[pet_count]] == pet_types[2]):
        number_of_birds = number_of_birds + 1
#
print (pet_owners[0], 'and', pet_owners[1], 'have:')
print(number_of_dogs, 'animals of type', pet_types[0])
print(number_of_cats, 'animals of type', pet_types[1])
print(number_of_birds, 'animals of type', pet_types[2])

print('Their names are:')
for pet_count in range (0, number_of_pets) :
    print(pet_names[pet_count], 'a', pets[pet_names[pet_count]])
```

Ln: 26 Col: 12

Results for Lists, Tuples, and Dictionaries Example

A screenshot of a Python 3.6.3 Shell window. The window has a title bar with three colored circles (red, yellow, green) on the left and the text "Python 3.6.3 Shell" on the right. The main area contains the following text: "Python 3.6.3 (v3.6.3:2c5fed86e0, Oct 3 2017, 00:32:08)", "[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin", "Type \"copyright\", \"credits\" or \"license()\" for more information.", ">>>", "RESTART: /Volumes/raid_blk/data/python_class/code_examples/list_tuples_dictionaries_example.py", "Alice and Bob have:", "2 animals of type Dog", "2 animals of type Cat", "0 animals of type Bird", "Their names are:", "Ralph a Dog", "Tom a Cat", "Blackie a Dog", "Boots a Cat", ">>>". The bottom right corner of the window shows "Ln: 15 Col: 4".

```
Python 3.6.3 (v3.6.3:2c5fed86e0, Oct 3 2017, 00:32:08)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: /Volumes/raid_blk/data/python_class/code_examples/list_tuples_dictionaries_example.py
Alice and Bob have:
2 animals of type Dog
2 animals of type Cat
0 animals of type Bird
Their names are:
Ralph a Dog
Tom a Cat
Blackie a Dog
Boots a Cat
>>>
```

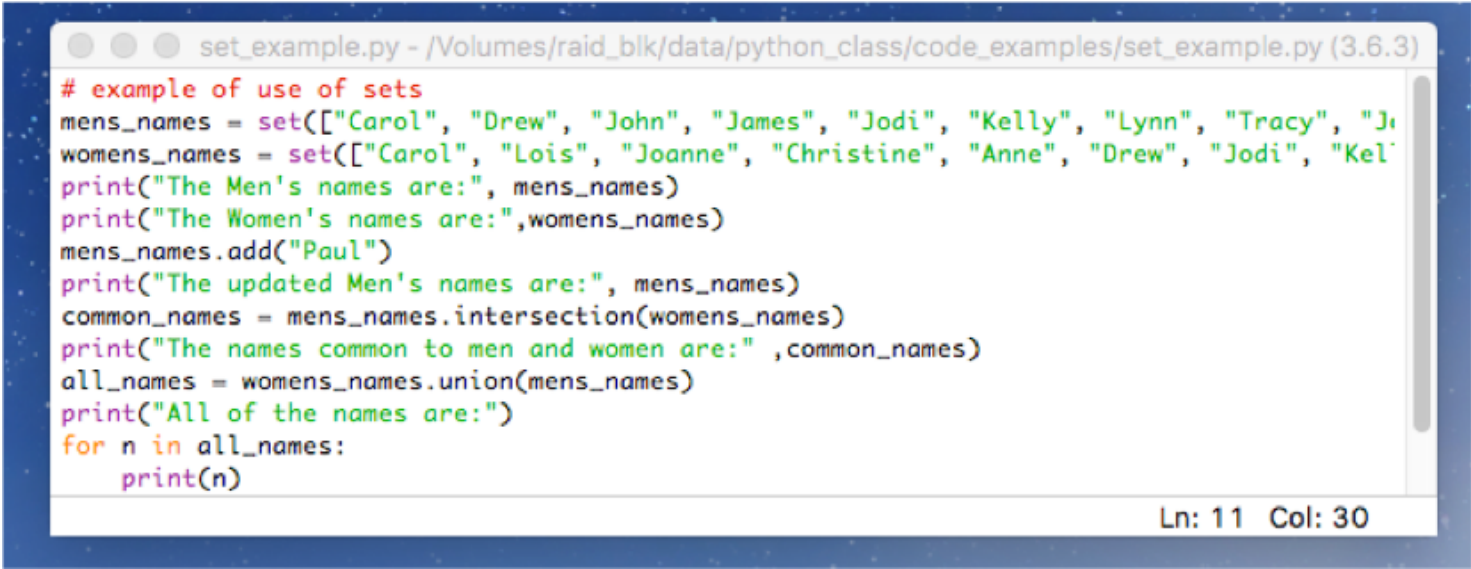

Sets

- Sets are a mathematical construct that is useful to categorize things
- There is a whole branch of Mathematics called Set Theory
- Python 3 supports sets by using {element1, element2, element3,...}
- Once we have one or more sets we can do operations on the set or sets
- Examples of Sets are:
 - names
 - cities
 - nations
 - capitals of states
 - prime numbers
 - even numbers

Set Operations

- `add(element)`
 - add an element to a set
- `clear()`
 - clear all elements from a set
- `copy`
 - create a shadow copy of a set
- `difference()`
 - show the differences between two set
- `difference_update()`
 - removes all elements of another set from this set
- `discard(element)`
 - remove an element from a set (if it doesn't exist → no error message)
- `remove(element)`
 - remove an element from a set (if it doesn't exist → `Keyerror` message)
- `union(set)`
 - create a new set that includes elements of two sets
- `intersection(set)`
 - A set with all the elements which are common to both sets
- `isdisjoint()`
 - returns `True` if two sets have a no common elements (null intersection)
- `issubset()`
 - `x.issubset(y)` returns `True`, if all elements in `x` are in `y` (`x` is a subset of `y`)
- `issuperset()`
 - `x.issuperset(y)` returns `True`, if `x` contains at least all elements in `y` (`x` is a superset of `y`)
- `pop()`
 - `pop()` removes and returns a random set element

Example Code for Set Operations



```
set_example.py - /Volumes/raid_blk/data/python_class/code_examples/set_example.py (3.6.3)
# example of use of sets
mens_names = set(["Carol", "Drew", "John", "James", "Jodi", "Kelly", "Lynn", "Tracy", "Jodi"])
womens_names = set(["Carol", "Lois", "Joanne", "Christine", "Anne", "Drew", "Jodi", "Kelly"])
print("The Men's names are:", mens_names)
print("The Women's names are:", womens_names)
mens_names.add("Paul")
print("The updated Men's names are:", mens_names)
common_names = mens_names.intersection(womens_names)
print("The names common to men and women are:", common_names)
all_names = womens_names.union(mens_names)
print("All of the names are:")
for n in all_names:
    print(n)
```

Ln: 11 Col: 30

Results of Example Code for Sets

```
Python 3.6.3 Shell
Python 3.6.3 (v3.6.3:2c5fed86e0, Oct  3 2017, 00:32:08)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Volumes/raid_blk/data/python_class/code_examples/set_example.py =
The Men's names are: {'Kelly', 'Jodi', 'Drew', 'Tracy', 'Frank', 'Lynn', 'Joe', 'Carol', 'Bill', 'John', 'James'}
The Women's names are: {'Kelly', 'Anne', 'Tracy', 'Joanne', 'Christine', 'Lois', 'Olivia', 'Lynn', 'Carol', 'Drew', 'Jodi'}
The updated Men's names are: {'Kelly', 'Jodi', 'Drew', 'Tracy', 'Paul', 'Frank', 'Lynn', 'Joe', 'Carol', 'Bill', 'John', 'James'}
The names common to men and women are: {'Kelly', 'Tracy', 'Lynn', 'Carol', 'Drew', 'Jodi'}
All of the names are:
Anne
Tracy
Paul
Joanne
Frank
Christine
Lois
Lynn
Carol
John
Jodi
James
Kelly
Olivia
Joe
Bill
Drew
>>>
```

Ln: 28 Col: 4

Running Python from the command line: Linux

- First:
 - Make sure the file is executable: `chmod +x script.py`
 - Use a shebang to let the kernel know what interpreter to use. The top line of the script should read:
 - `#!/usr/bin/python`
 - If the path to the python interpreter is wrong we get an obtuse error message
 - 'No such file or directory'
 - This assumes that your script will run with the default python. If you need a specific version, just specify in the shebang:
 - `#!/usr/bin/python2.7`
- Now you can type:
 - `./script.py` if the script is in your current directory, or:
 - `script.py` if the location of the script happens to be in your PATH, or:
`path/to/script.py` otherwise.
- Some programs require elevated status (i.e., run as root)
 - `sudo ./script.py` or
 - `sudo python3 script.py`

Running Python from the command line:

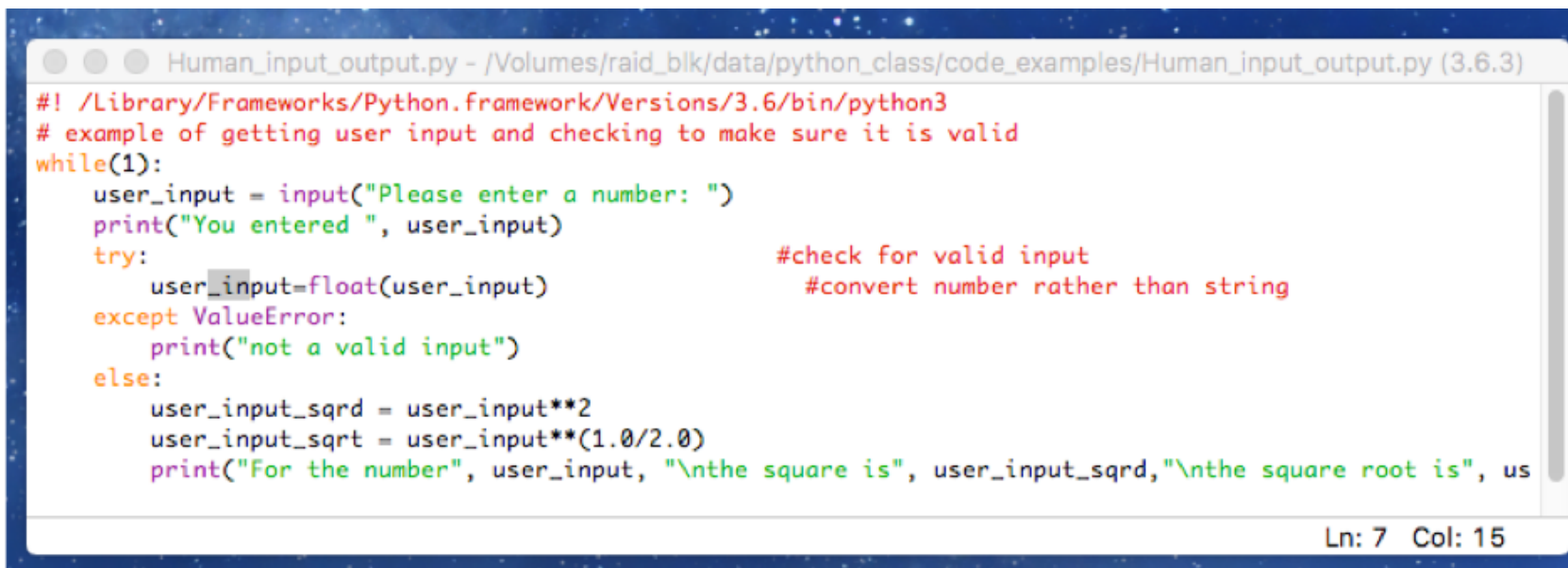
Windows and Mac

- Open a Terminal Window (Mac) or Command Window (Windows)
- change directory to location of the python script
- type: `python3 script.py`
- For the Mac, adding
 - `#!/Library/Frameworks/Python.framework/Versions/3.6/bin/python3`
 - to the first line does work
 - `chmod +x script.py`
 - run the program by typing: `./script.py`

Human Inputs

- Sometimes we want to get input from the user
- We must prompt the user for the type of input requested
- Then we must collect the input
- Since Python stores variables implicitly, we often have to convert the representation of the input to the correct format
 - `int()` for integer numbers
 - `float()` for floating point numbers
 - `long()` for long integer numbers
 - `str()` for strings

Example of Human Input

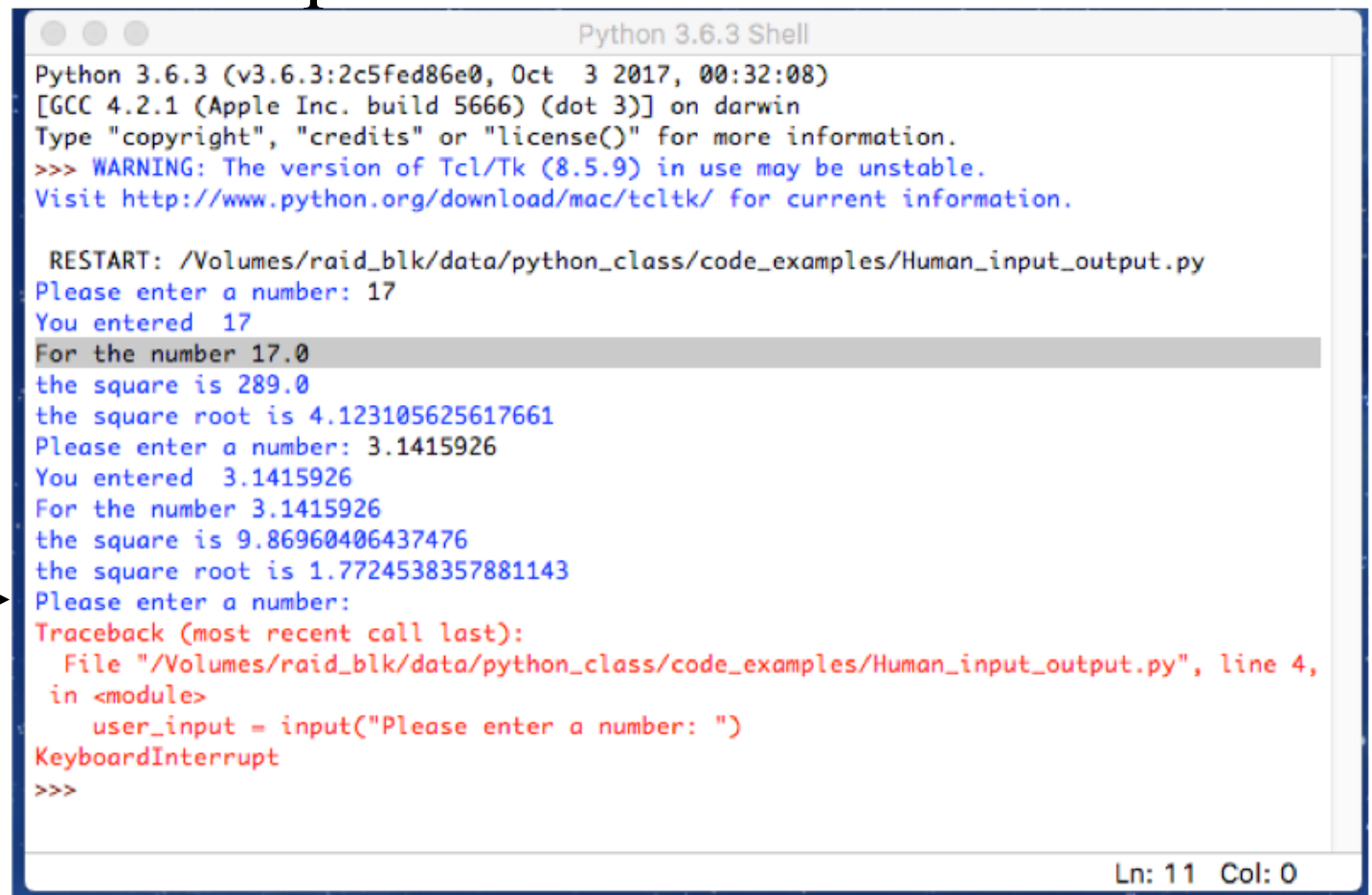


```
Human_input_output.py - /Volumes/raid_blk/data/python_class/code_examples/Human_input_output.py (3.6.3)
#!/Library/Frameworks/Python.framework/Versions/3.6/bin/python3
# example of getting user input and checking to make sure it is valid
while(1):
    user_input = input("Please enter a number: ")
    print("You entered ", user_input)
    try:
        user_input=float(user_input)
        #check for valid input
        #convert number rather than string
    except ValueError:
        print("not a valid input")
    else:
        user_input_sqrd = user_input**2
        user_input_sqrt = user_input**(1.0/2.0)
        print("For the number", user_input, "\nthe square is", user_input_sqrd, "\nthe square root is", us
```

Ln: 7 Col: 15

Results of Human Input

Enter
Control-C
to exit program



```
Python 3.6.3 Shell
Python 3.6.3 (v3.6.3:2c5fed86e0, Oct 3 2017, 00:32:08)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

RESTART: /Volumes/raid_blk/data/python_class/code_examples/Human_input_output.py
Please enter a number: 17
You entered 17
For the number 17.0
the square is 289.0
the square root is 4.123105625617661
Please enter a number: 3.1415926
You entered 3.1415926
For the number 3.1415926
the square is 9.86960406437476
the square root is 1.7724538357881143
Please enter a number:
Traceback (most recent call last):
  File "/Volumes/raid_blk/data/python_class/code_examples/Human_input_output.py", line 4,
    in <module>
      user_input = input("Please enter a number: ")
KeyboardInterrupt
>>>
```

Ln: 11 Col: 0

File I/O

- `output = open(r'C:\spam', 'w')` Create output file ('w' means write)
- `input = open('data', 'r')` Create input file ('r' means read)
- `input = open('data')` Same as prior line ('r' is the default)
- `aString = input.read()` Read entire file into a single string
- `aString = input.read(N)` Read up to next N characters (or bytes) into a string
- `aString = input.readline()` Read next line (including `\n` newline) into a string
- `aList = input.readlines()` Read entire file into list of line strings (with `\n`)
- `output.write(aString)` Write a string of characters/bytes into file
- `output.writelines(aList)` Write all line strings in a list into file

File I/O (con't)

- `output.close()`
- `output.flush()`
- `anyFile.seek(N)`
- `for line in open('data'):` use line
- `open('f.txt', encoding='latin-1')`
- `open('f.bin', 'rb')`
- `codecs.open('f.txt', encoding='utf8')`
- `open('f.bin', 'rb')`

Manual close
(done for you when file is collected)

Flush output buffer to disk without closing

Change file position to offset N for next operation

File iterators read line by line

Python 3.X Unicode text file
(str strings)

Python 3.X bytes files (bytes strings)

Python 2.X Unicode text files
(unicode strings)

Python 2.X bytes files (str strings)

Exception coding

- When we have large programs, we may need to jump out of pieces of the program
- When we encounter an error, the program will terminate, i.e., stop running
- If we can capture the error we may be able to fix the problem and continue running the program
- Example: I build a calculator. I ask for input of a number. The person enters "z". The program crashes/terminates/stops running
- If I capture the wrong input, I can ask the user to enter the correct type of data

Exception Coding

- `try/except/else`
- `try/except/finally`
- `raise`
- `assert`

try/except/else

- The try statement works as follows.
 - First, the try clause (the statement(s) between the try and except keywords) is executed.
 - If no exception occurs, the except clause is skipped and execution of the try statement is finished.
 - If an exception occurs during execution of the try clause, the rest of the clause is skipped.
 - Then if its type matches the exception named after the except keyword, the except clause is executed, and then execution continues after the try statement.
 - If an exception occurs which does not match the exception named in the except clause, it is passed on to outer try statements;
- If no handler is found, it is an unhandled exception and execution stops with a message as shown above.
- A try statement may have more than one except clause, to specify handlers for different exceptions.
- At most one handler will be executed. Handlers only handle exceptions that occur in the corresponding try clause, not in other handlers of the same try statement.

try/except/finally

- This set of statements works similar to the try/except/else
- The "finally" statement is always run for no exceptions and handled exceptions
- It's best use it to make the code clearer
- It is not required

raise

- When testing code, it may not be possible to determine how an exception will occur
- The raise statement is used to force an exception for testing
- It should be removed after testing
- It can also be added around a statement the only get executed when the debug mode is enabled
 - See python documentation for debug mode
 - <https://docs.python.org/3.2/library/pdb.html>

assert

- The goal of an assertion in Python is to inform developers about unrecoverable errors in a program.
- Python's assert statement is a debugging aid, not a mechanism for handling run-time errors.
- The goal of using assertions is to let developers find the likely root cause of a bug more quickly.
- An assertion error should never be raised unless there's a bug in your program
- Basically the approach is to make (force) an error to test whether or not the error is handled correctly

Example code for exceptions

```
exceptions.py - /Users/joe/Documents/exceptions.py (3.6.3)
"""
This code presents examples for several types of exception handling.
The goal is to keep the code running smoothly in the presence of problems
like poor user input
unknown errors
"""

# try except example
for i in range(1,3):
    try:
        y= input("Please enter a number: ")
        x = int(y)
        print("thank you, that was a valid number of value: ",x)
    except ValueError:
        print("Oops! That was not a valid number. You entered \" ", y, "\" Try again...")
        print(" ")

# try except finally example
for i in range(1,3):
    x= input("Please enter the numerator : ")
    x=int(x)
    y= input("Please enter the denominator: ")
    y=int(y)
    try:
        result = x/y
        print("The result of the division is ",result)
    except ZeroDivisionError:
        print("Oops! dividing by zero Try again...")
    finally:
        print("finally runs whether there is an exception or no exception")
    print(" ")

# assert example
for i in range(1,2):
    try:
        y= input("Please enter a number: ")
        x = int(y)
        assert int("a")
        print("thank you, that was a valid number of value: ",x)
    except ValueError:
        print("Oops! That was not a valid number. You entered \" ", y, "\" Try again...")
        print(" ")

# try except finally example with raise
for i in range(1,2):
    x= input("Please enter the numerator : ")
    x=int(x)
    y= input("Please enter the denominator: ")
    y=int(y)
    try:
        result = x/y
        raise ZeroDivisionError
        print("The result of the division is ",result)
    except ZeroDivisionError:
        print("Oops! dividing by zero Try again...")
    finally:
        print("finally runs whether there is an exception or no exception")
    print(" ")

# raise example without being handled
print("now we raise an exception without it being handled")
raise ZeroDivisionError
```

Ln: 61 Col: 59

Running of example code

```
Python 3.6.3 Shell
Python 3.6.3 (v3.6.3:2c5fed86e0, Oct 3 2017, 00:32:08)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/joe/Documents/exceptions.py =====
Please enter a number: d
Oops! That was not a valid number. You entered " d " Try again...

Please enter a number: 2
thank you, that was a valid number of value: 2

Please enter the numerator : 1
Please enter the denominator: 2
The result of the division is 0.5
finally runs whether there is an exception or no exception

Please enter the numerator : 3
Please enter the denominator: 4
The result of the division is 0.75
finally runs whether there is an exception or no exception

Please enter a number: 1
Oops! That was not a valid number. You entered " 1 " Try again...

Please enter the numerator : 1
Please enter the denominator: 2
Oops! dividing by zero Try again...
finally runs whether there is an exception or no exception

now we raise an exception without it being handled
Traceback (most recent call last):
  File "/Users/joe/Documents/exceptions.py", line 62, in <module>
    raise ZeroDivisionError
ZeroDivisionError
>>> |
```

Ln: 35 Col: 4

Coding Mistakes (Gotchas)/ Best Practices

- Start all code in column 1 unless it should be indented
- Use consist indents, i.e., the same number of spaces for each indent
 - Don't use tabs; make sure you editor converts tabs to spaces
 - Don't switch editors, especially between OSs, this can cause inconsistent idents
- Don't forget to use a colon, :, at the end of statements that need it
 - if, while, for, etc.
- Not a mistake but a best practice: Make a lot of comments
 - Make sure that you can understand you code, 1 week, 1 month or 5 years from now.
 - What is obvious today is not obvious tomorrow

More Coding Mistakes (Gotchas)/ Best Practices

- Don't end statements with a semi-colon, ;
 - It is permitted to allow multiple statements on one line but not recommended
- Don't enclose tests in parentheses, i.e.,
 - Use `if test:`
 - Don't use `if(test):`
 - It is not needed but may make a compound test more understandable
- Don't use { and }
- Don't reuse variables within the code
- Make variable names explicit
 - Example: for a robot with a sensor that can turn +/-90 degrees in 5 degree steps
 - Use `turret_position` rather than `i` for the index
- When calling functions with no arguments use `function()` not `function`
-

Debugging Programs

- It is rare to write a program that works when first written
- If someone says there are no bugs in their software, don't believe them
- There are usually mistakes (called bugs) in the program
- Sometimes the interpreter will flag them, mostly syntax errors
- Other times, it will be necessary to study the code carefully to find the errors
- Large programs are rarely bug free
 - Program robustness is defined by the time to find the next bug
 - It ideally should be years but sometimes it is days
- Debugging a program is a skill/art

Areas to examine in debugging programs

- Range of loops
- Tests for if, elif, else, while,
- Valid input and outputs
 - Screen prompt: Enter a number between 0 and 9:
 - User enters: A instead of a number, etc
- missing [] : { } ()
- improper indentation

Home Work

- The example for lists, tuples and Dictionaries is not written very well:
 - Why?
 - How could it be fixed
- Debug `non_working_example.py` program
- What did you learn about debugging?
- Write a program to read the file `stock_data.csv` and format the output to generate a report on the data in the file
- Expand the previous program to delete lines 3, 5 and 11 and write a new file called `revised_stock_data.csv`