Introduction to Python 3

Week 2

Questions on last week?

Functions

- When writing programs, if we use the same code multiple times, it is more efficient to make it a function that has an arbitrary input and output
- For example, if we had a set of temperature reading every day for a year in degrees Fahrenheit and wanted to convert them to degrees Celsius, we could write a f_to_c function
  - This is clearly a trivial example

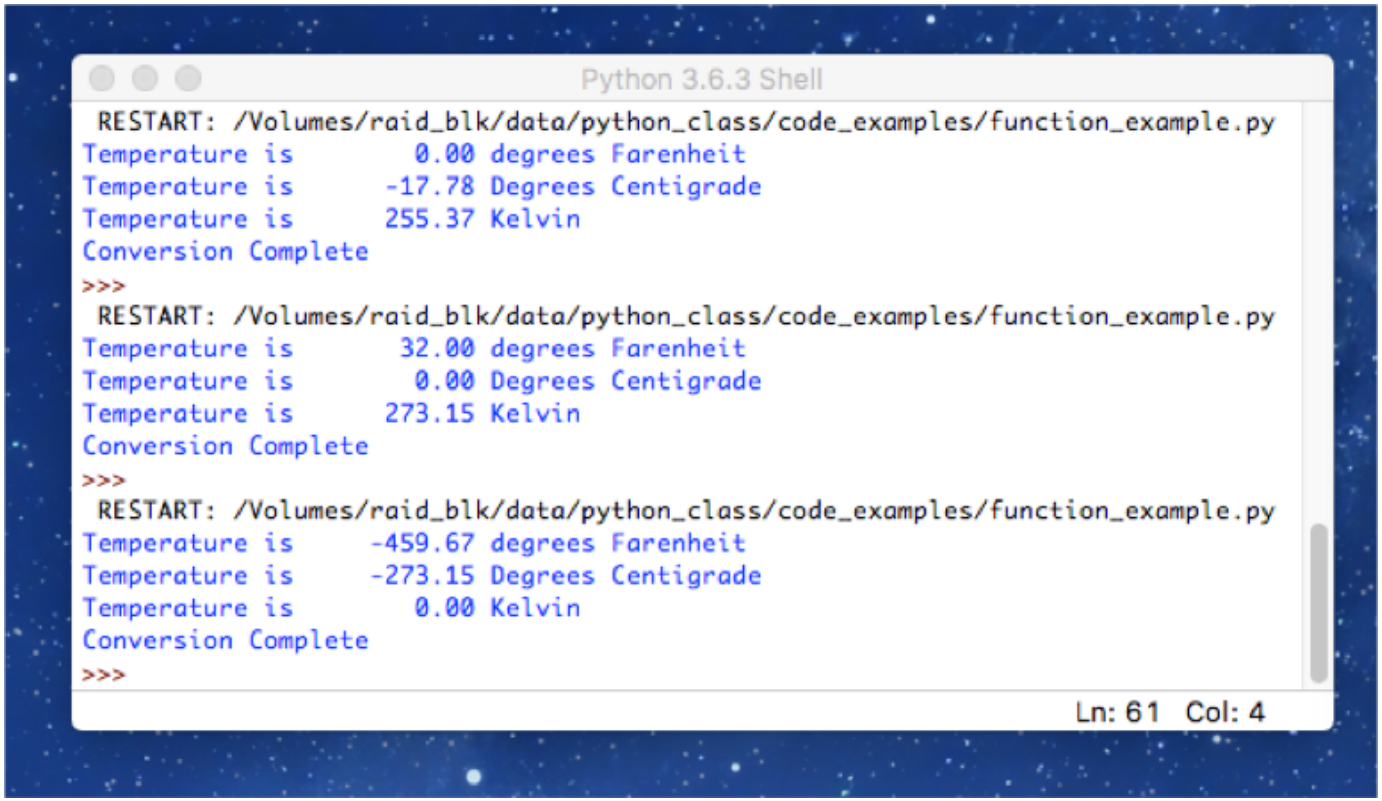
Example Function

function_example.py - /Volumes/raid_blk/data/python_class/code_examples/function_example.py (3.6.3)

```
# example for a function
def f_to_c(f):
    c = (f-32.0)*(5.0/9.0)
    return c

def c_to_f (c):
    f = c*9.0/5.0+32.0
    return f

def c_to_k(c):
    k = c + 273.15
    return k

def k_to_c(k):
    c = k - 273.15
    return c

def f_to_k(f):
    k = (f-32.0)*(5.0/9.0) + 273.15
    return k

def k_to_f(k):
    f = (k-273.15)*(9.0/5.0)+32.0
    return f


temperature = 0.0
scale = "K"
if (scale == "F"):
    print("Temperature is ", "%10.2f"% temperature, "degrees Farenheit")
    print("Temperature is ", "%10.2f"% f_to_c(temperature), "Degrees Centigrade")
    print("Temperature is ", "%10.2f"% f_to_k(temperature), "Kelvin")
elif(scale == "C"):
    print("Temperature is ", "%10.2f"% c_to_f(temperature), "degrees Farenheit")
    print("Temperature is ", "%10.2f"% temperature, "Degrees Centigrade")
    print("Temperature is ", "%10.2f"% c_to_k(temperature), "Kelvin")
elif (scale == "K"):
    print("Temperature is ", "%10.2f"% k_to_f(temperature), "degrees Farenheit")
    print("Temperature is ", "%10.2f"% k_to_c(temperature), "Degrees Centigrade")
    print("Temperature is ", "%10.2f"% temperature, "Kelvin")
else:
    print("Wrong scale specified")
print("Conversion Complete")
```

Ln: 27 Col: 0

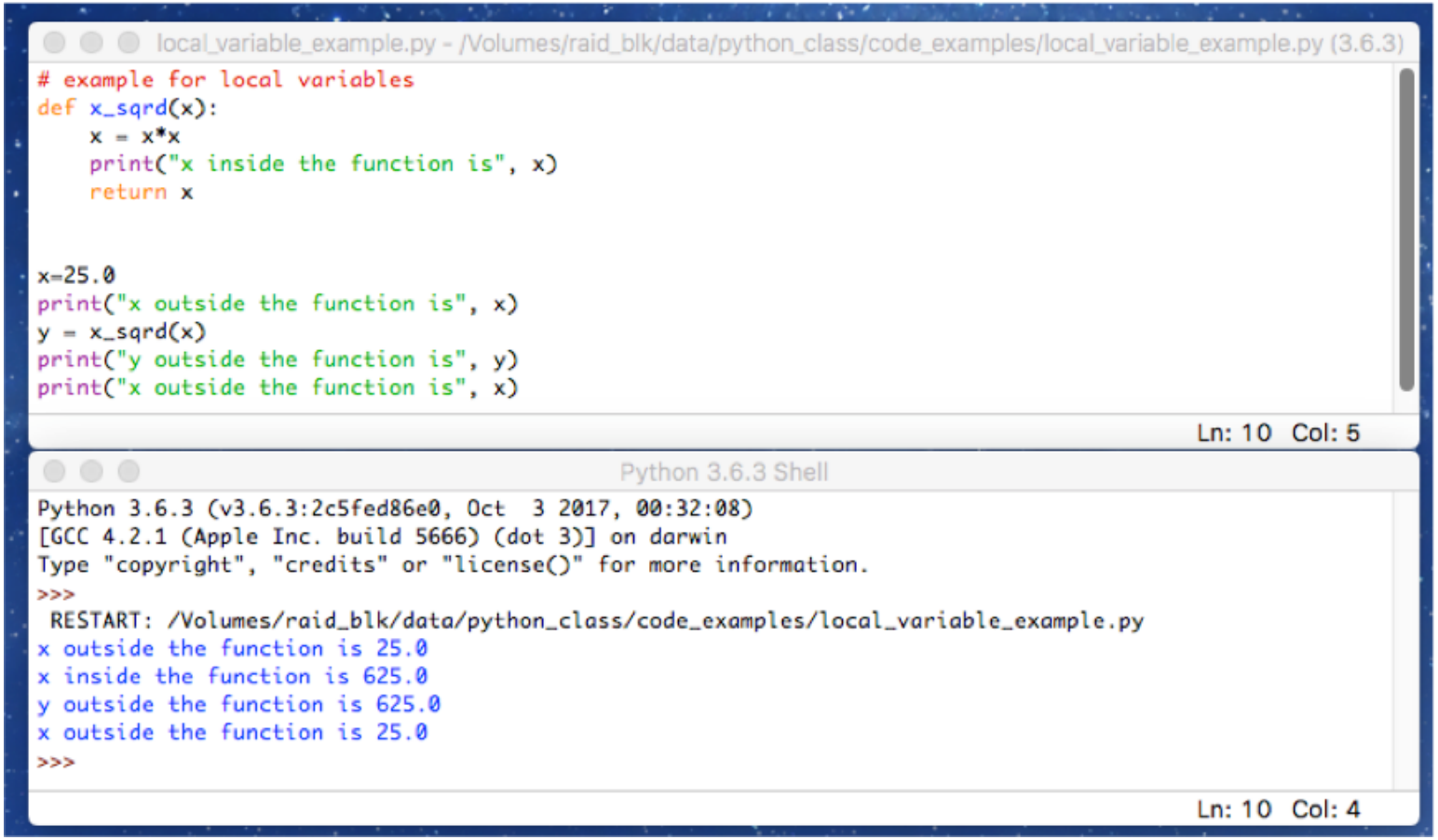
Output from example function



Variables are implicitly local unless declared as global

- Variable names are local to a program and a function
- The names of variables inside a function are not related to those outside the function
- When a function is called only the values are passed to the function
  - The names of the variables are not passed to the function
- Any variables used inside a function can not be referenced outside a function
  - The value of a variable (but not its name) can be passed in the return function
- Even though variables inside and outside a function are unrelated, it is recommend that variable names not be reused

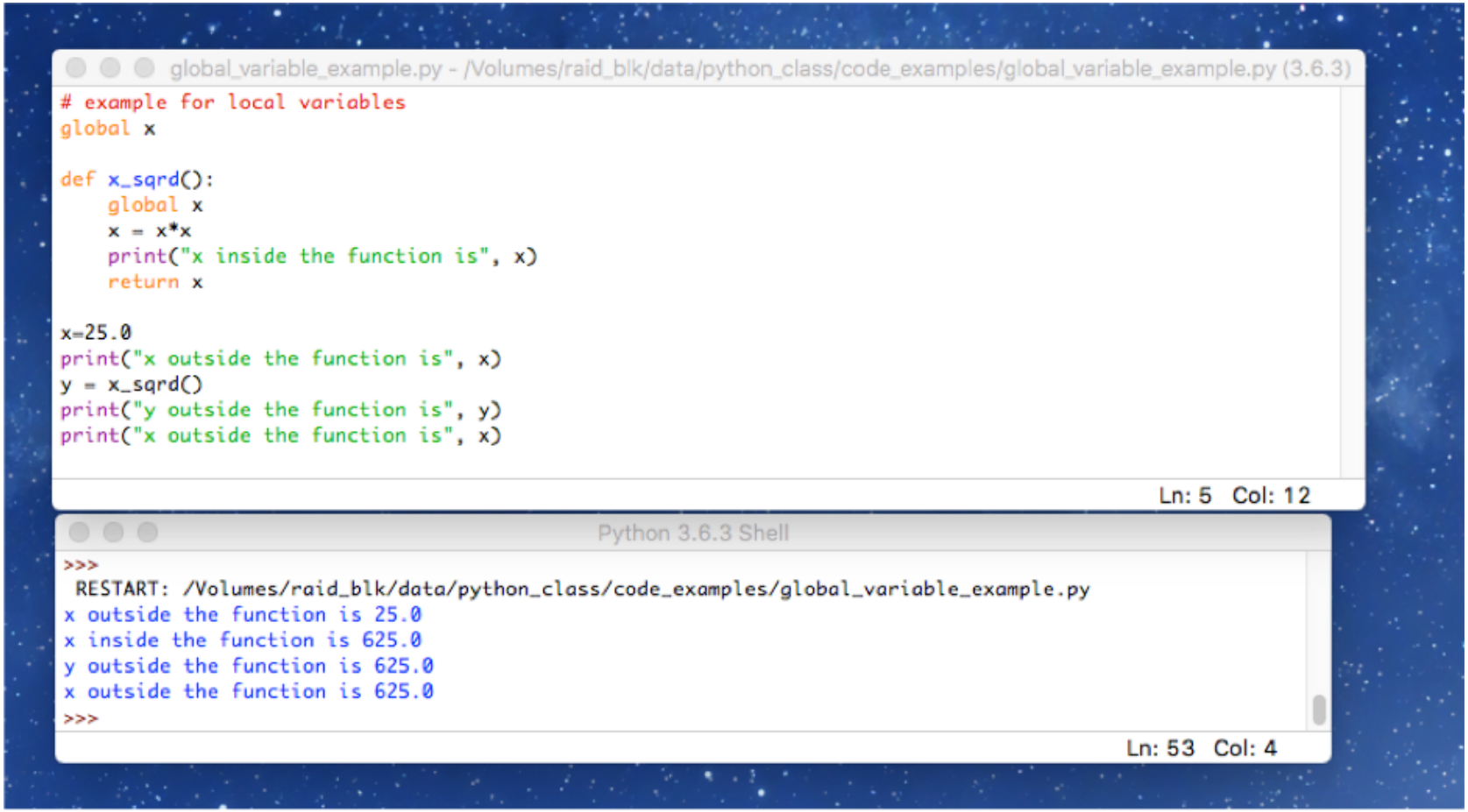
Local Variable Example



Global Variables

- A global variable is a variable known to all functions and can be used in any functions in the program
- It is not recommended to use global variables!
  - If a variable name is accidently reused or changed inside a function it can make debugging code very hard
  - It is better to pass data using arguments in a function and arguments in a return
- There are times when global variables are needed
  - Could not find any examples in Python
  - Unrelated to Python--- DSP programming C; had to pass large amount of data to a function via a global to make the code run correctly
- We will discuss global variables in more detail in the last week

Global Example



Libraries

- When there is a set of code that many people write and/or use, languages will be extended with functions that are then called libraries
- Libraries are called just like functions
- But they have to be included using the “import” statement before they are used.

Some common built-in libraries

- time
- math
- random
- os
- sys

- For more information on libraries see:
  - https://docs.python.org/3/library/index.html

Some Useful Libraries

- wxPython. A gui toolkit for python.
- NumPy. It provides some advance math functionalities to python.
- SciPy. It is a library of algorithms and mathematical tools for python
- matplotlib. A numerical plotting library. It is very useful for any data scientist or any data analyzer.
- Pygame. This library will help you achieve your goal of 2d game development. It has other uses, also.
- Scapy. A packet sniffer and analyzer for python made in python.

Getting libraries from the internet

- If you know the name of the library you can often download it directly using the pip command in a terminal window
- For Python 3, the command is pip3
- It should have been installed with Python
- We will show it use in a few slides

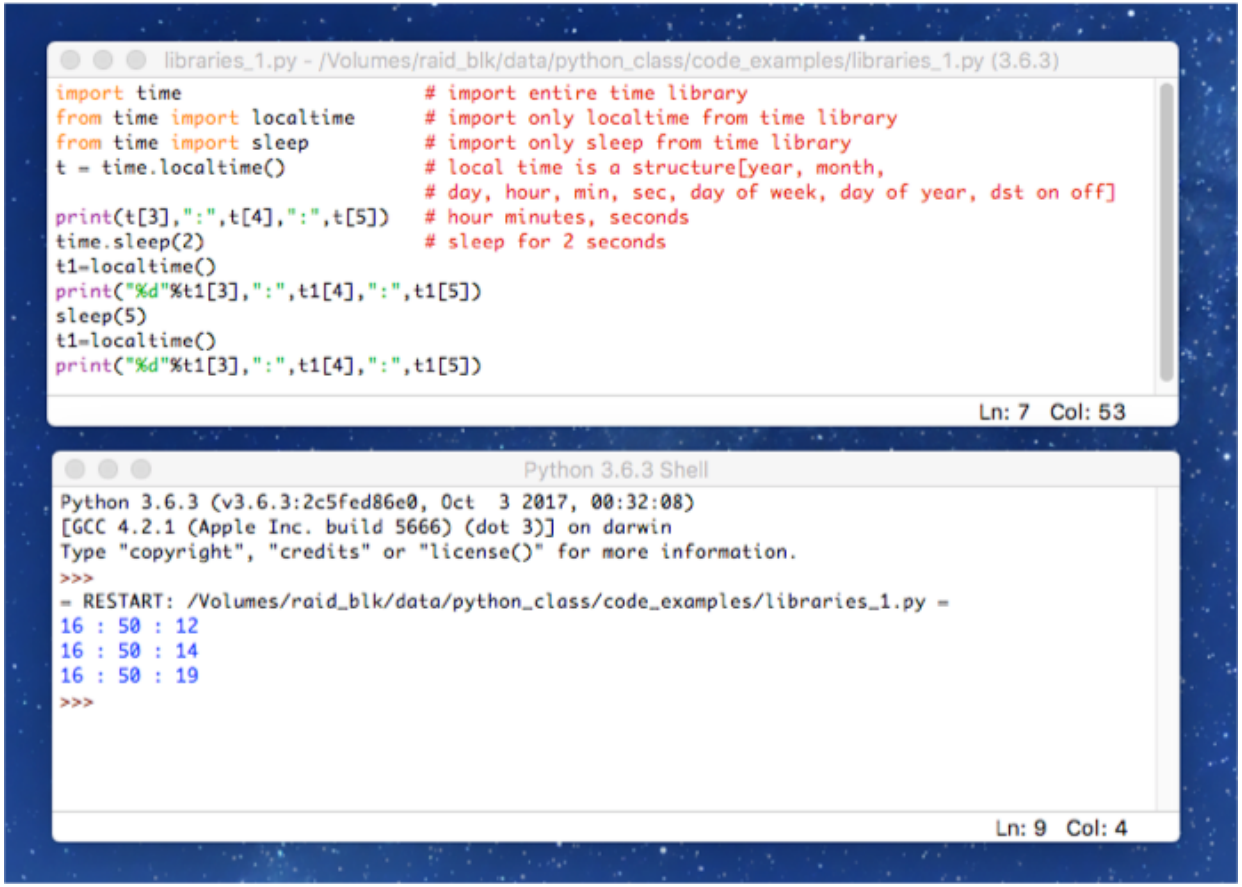
Importing libraries

- We import libraries with the import command
- The import command should be a the start of the program
  - import library_name
- This will install all functions of the library
- If code size is important, only the functions that you want to use can be installed
  - from library_name import function_name1 function_name2 …

Time Library

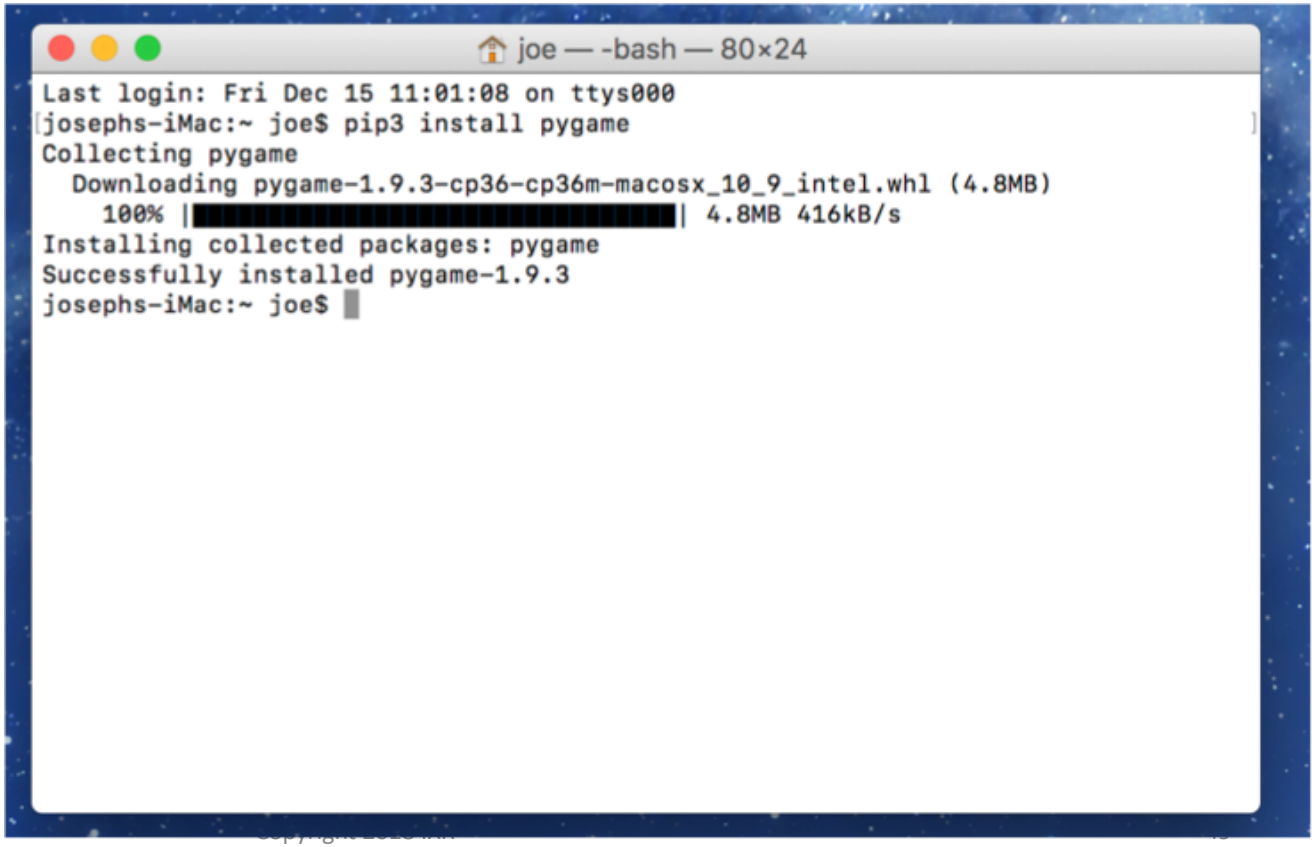
- The time library has many functions
- See https://docs.python.org/3/library/time.html
- Some of the useful functions are:
  - time.sleep(time_in_seconds)
  - time.localtime()
  - time.time()
    - On Windows and Unix Systems, return the time in seconds January 1, 1970, 00:00:00 (UTC) with leap seconds are not counted towards the time in seconds. This is commonly referred to as Unix time.
  - time.timezone()
  - time.daylight()

Example use of time library

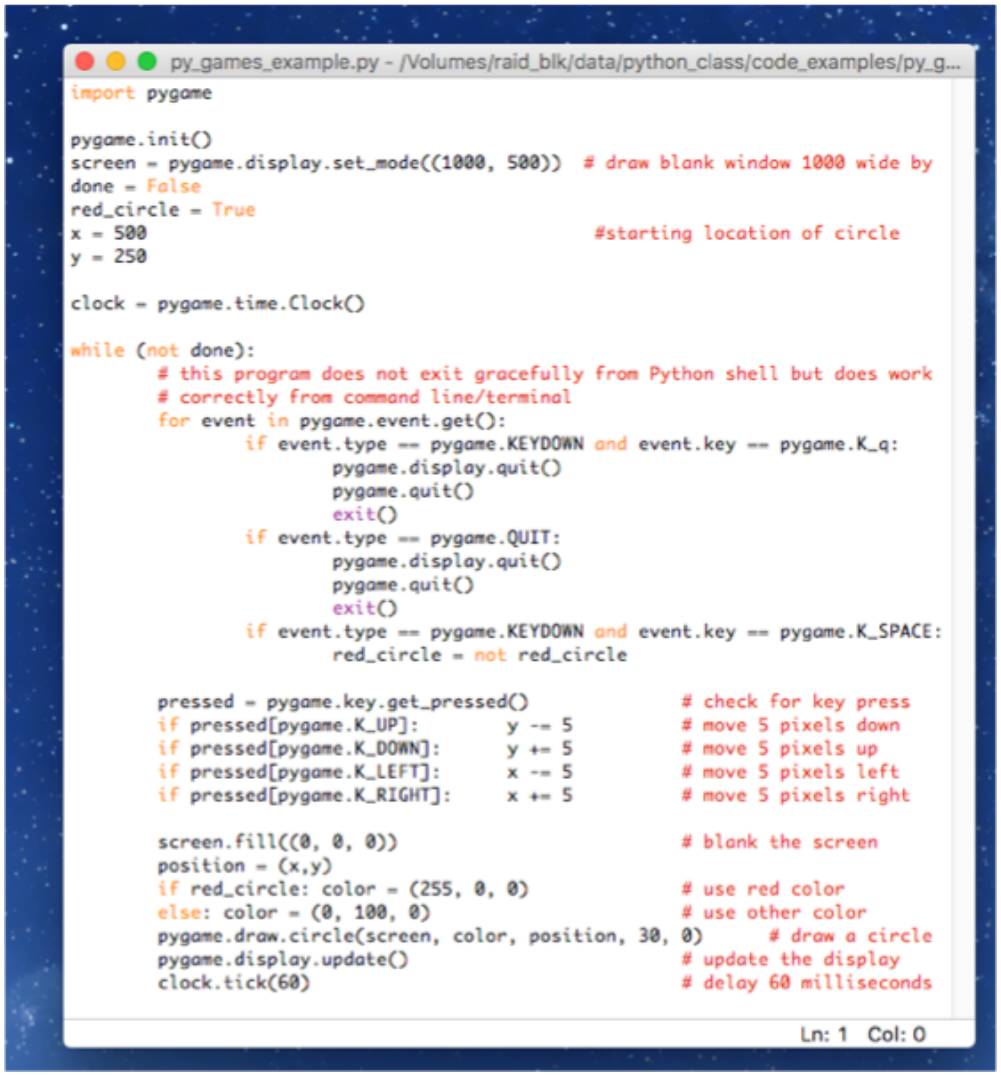


pygame library

- Install pygame from terminal window

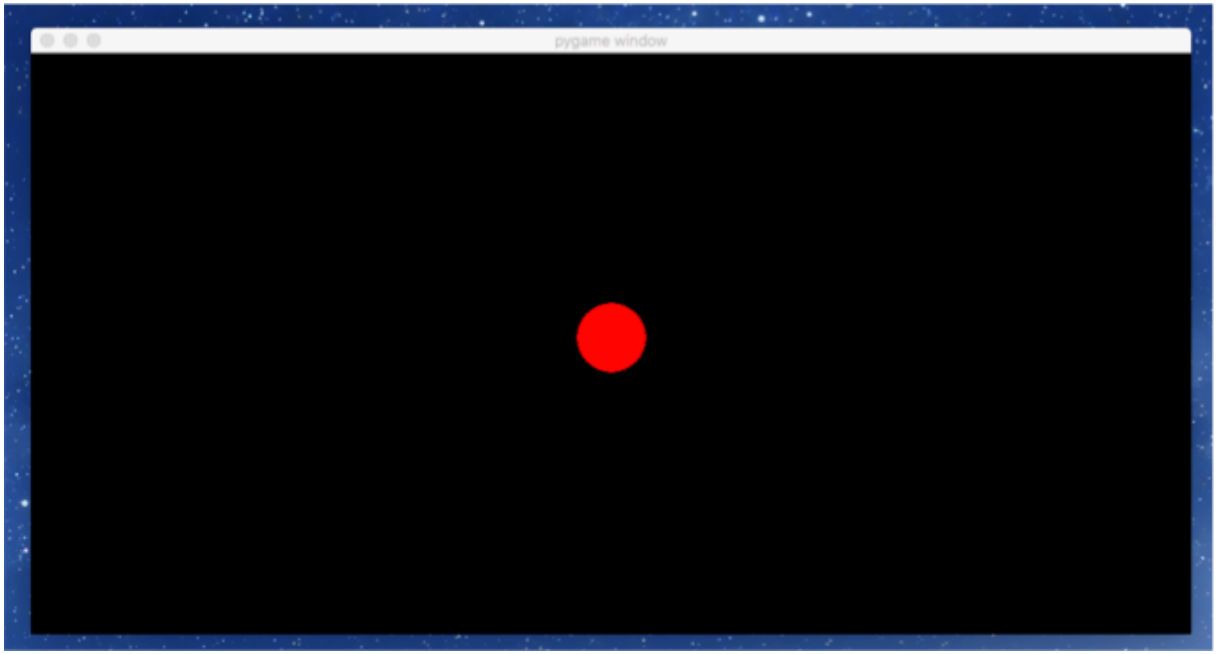


Example pygame Program



Program Running

- Circle can
  be moved with cursor



Printing

- Printing for Python 3 is different from Python 2
- We will use the Python 3 formatting conventions
- Printing is necessary to get output/results from your program
- We saw examples of printing in previous examples
- These examples will be more detailed
- We can print text and variables on single and multiple lines

Formatting of Variables when printing

- %c character
- %s string conversion via str() prior to formatting
- %i signed decimal integer
- %d signed decimal integer
- %u unsigned decimal integer
- %o octal integer
- %x hexadecimal integer (lowercase letters)
- %X hexadecimal integer (UPPERcase letters)
- %e exponential notation (with lowercase 'e')
- %E exponential notation (with UPPERcase 'E')
- %f floating point real number
- %g the shorter of %f and %e
- %G the shorter of %f and %E

Printing examples

- print (“This is a line of text”)
- print (“This is a second line of text”)
- print (“This is a line of text\nThis is a second line of text”)

- The \n tells the program to start a new line
- A \ before a special command will print the command
- print("\\n") will print \n
- print("\n") will move to the next line for the next print statement

Some Common Formatting Expressions

- \n new line
- \' '
- \" "
- \f form feed
- \r carriage return (CR) move to beginning of same line
- \t tab

Formatting variables for Printing

- Print decimal variable with Z spaces and y digits after the decimal point
  - %Z.y**d**"%
- Print floating variable with Z spaces and y digits after the decimal point
  - %Z.y**f**"%
- Print octal variable with Z spaces and y digits after the octal point
  - %Z.y**o**"%
- Print binary variable with Z spaces and y digits after the binary point
  - %Z.y**b**"%
- Print hex variable with Z spaces and y digits after the hex point
  - %Z.y**X**"%

- **Note: use of red Z, blue y, and bold type is for clarity and not a part of the formatting**

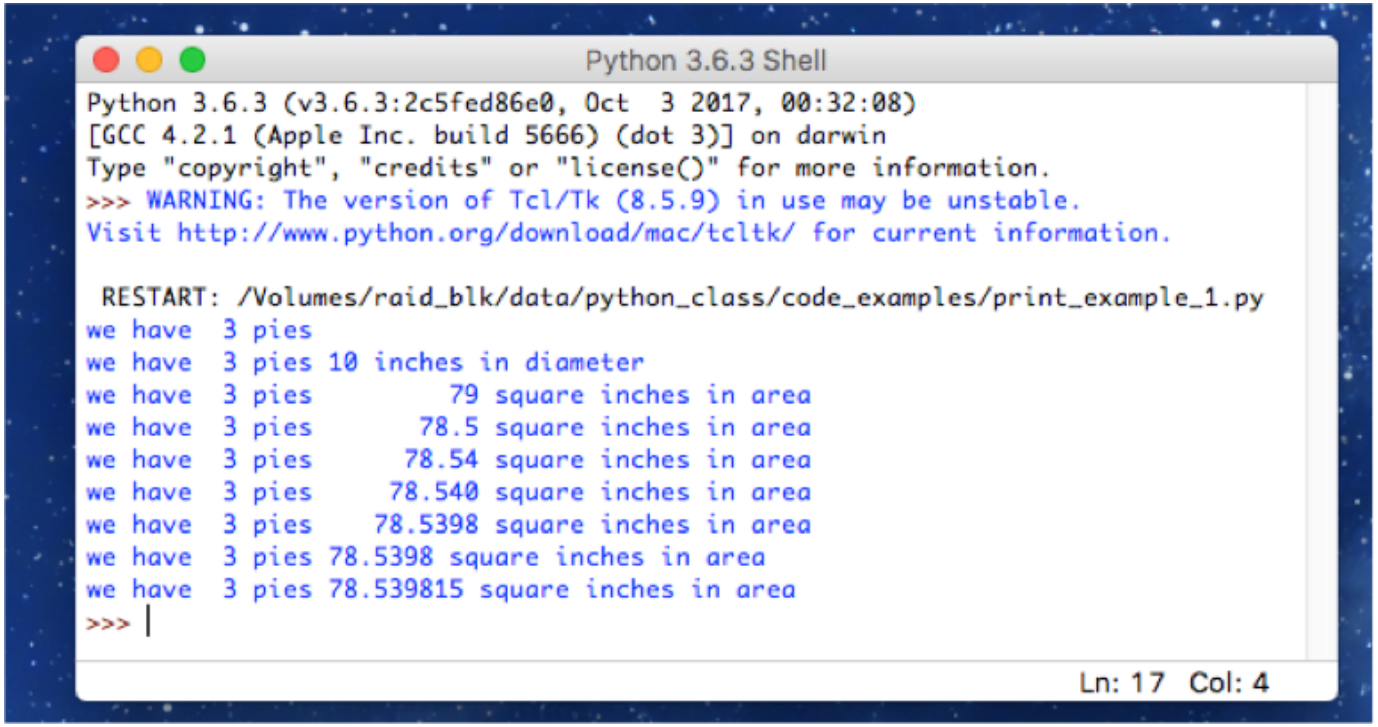
Print_example_1.py

print_example_1.py - /Volumes/raid_blk/data/python_class/code_examples/print_example_1.py (3.6.3)

```
number_of_pies=3
pie=3.1415926
diameter_of_pie=10
print ("we have ", number_of_pies, "pies")
print ("we have ", number_of_pies, "pies", diameter_of_pie, "inches in diameter")
area_of_pie =pie*(diameter_of_pie/2)**2
print ("we have ", number_of_pies, "pies", "%10.0f"% area_of_pie, "square inches in area")
print ("we have ", number_of_pies, "pies", "%10.1f"% area_of_pie, "square inches in area")
print ("we have ", number_of_pies, "pies", "%10.2f"% area_of_pie, "square inches in area")
print ("we have ", number_of_pies, "pies", "%10.3f"% area_of_pie, "square inches in area")
print ("we have ", number_of_pies, "pies", "%10.4f"% area_of_pie, "square inches in area")
print ("we have ", number_of_pies, "pies", "%5.4f"% area_of_pie, "square inches in area")
print ("we have ", number_of_pies, "pies", "%f"% area_of_pie, "square inches in area")
```

Ln: 12 Col: 46

Print_example_1.py



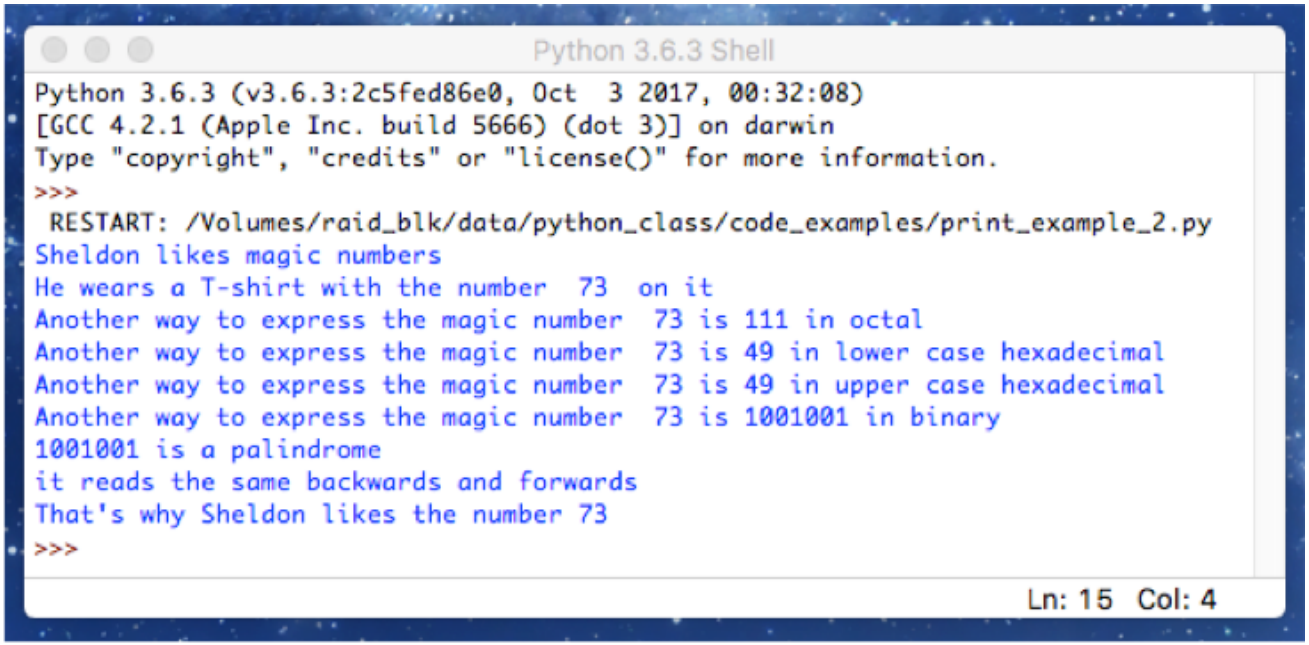
print_example_2.py

print_example_2.py - /Volumes/raid_blk/data/python_class/code_examples/print_example_2.py (3.6.3)

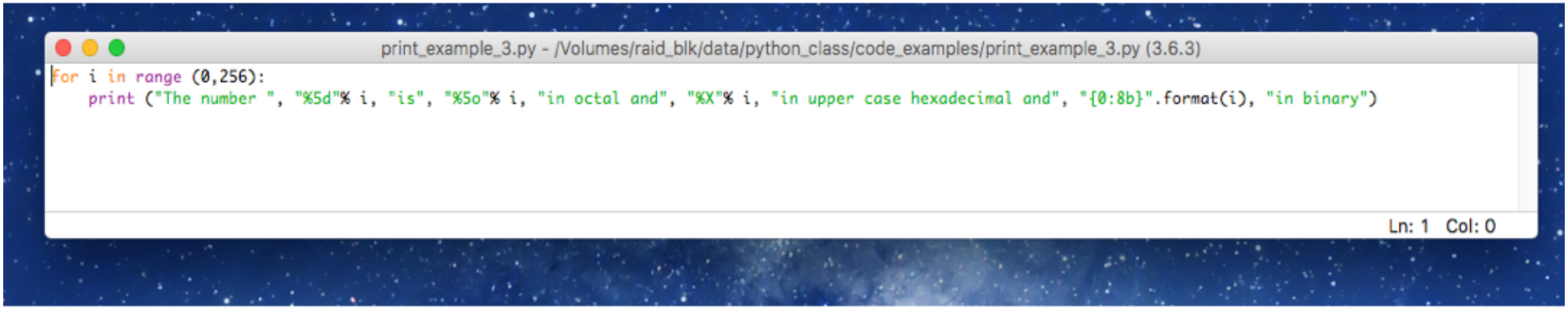
```
magic_number=73
print ("Sheldon likes magic numbers\nHe wears a T-shirt with the number ", magic_number, " on it")
print ("Another way to express the magic number ", magic_number, "is", "%o"% magic_number, "in octal")
print ("Another way to express the magic number ", magic_number, "is", "%x"% magic_number, "in lower case hexadecimal")
print ("Another way to express the magic number ", magic_number, "is", "%X"% magic_number, "in upper case hexadecimal")
print ("Another way to express the magic number ", magic_number, "is", "{0:b}".format(magic_number), "in binary")
print ("{0:b}".format(magic_number), "is a palindrome\nit reads the same backwards and forwards")
print ("That's why Sheldon likes the number", magic_number)
```

Ln: 1 Col: 0

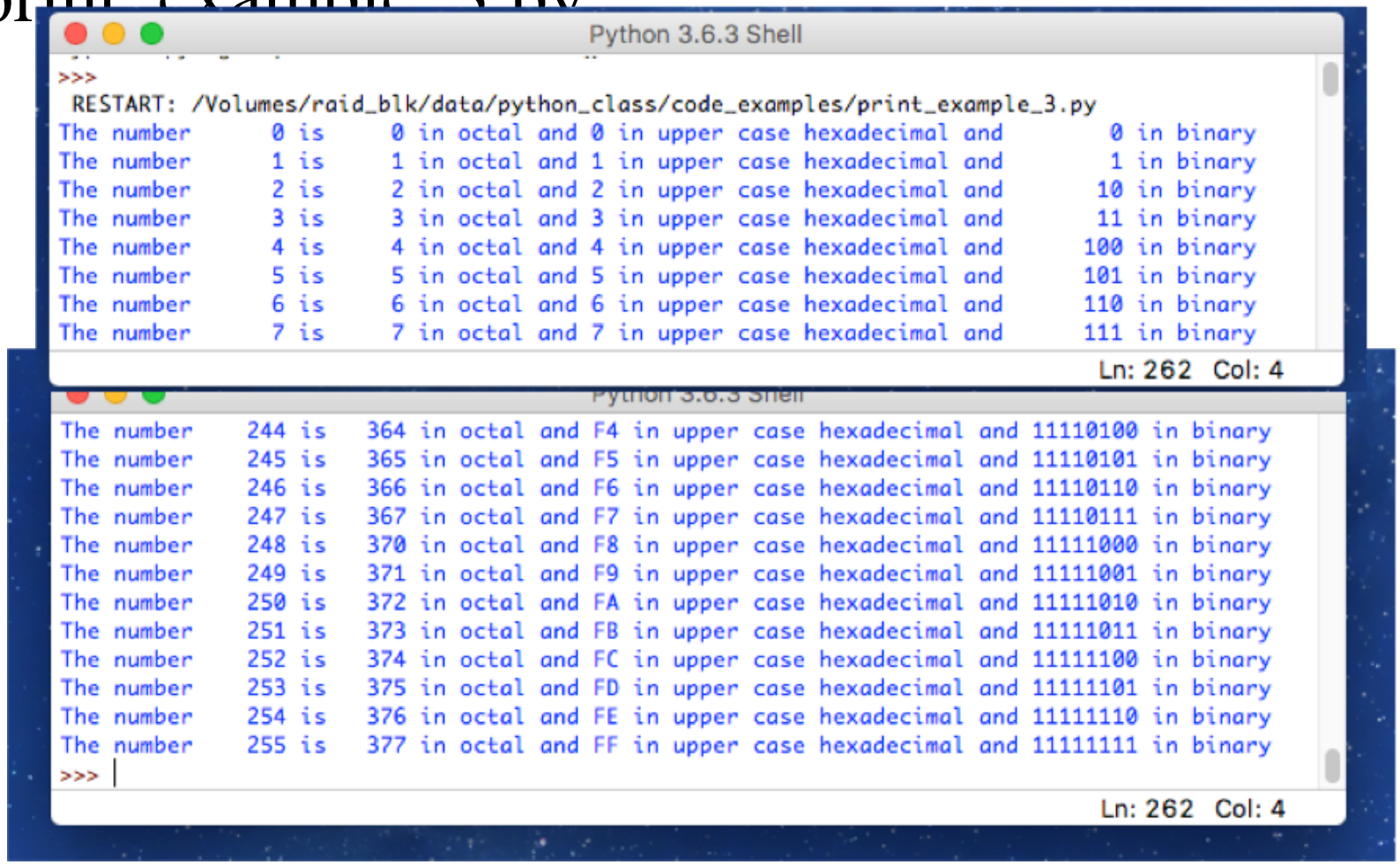
print_example_2.py



example_3.py



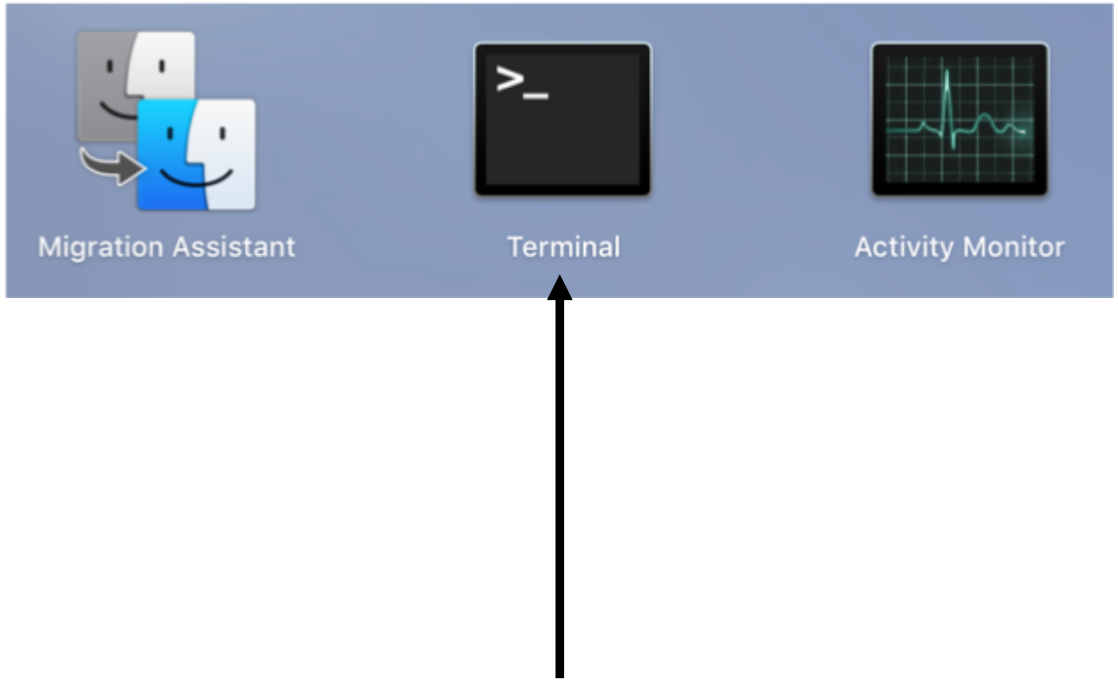
print_example_3.py



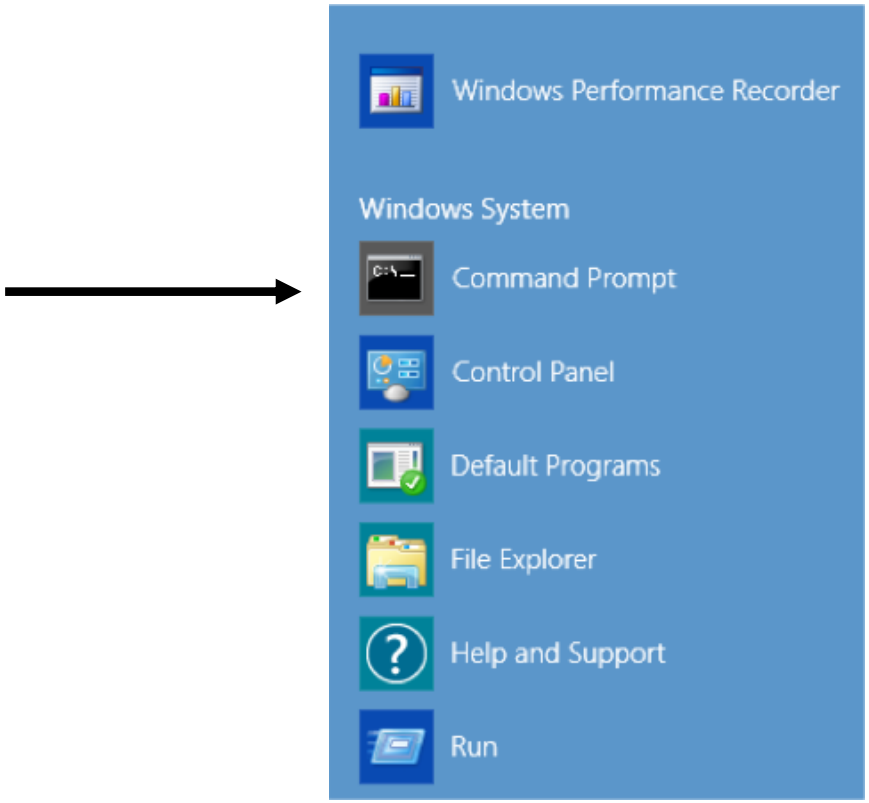
Calls to the OS

- Sometimes the program needs information from the operating system
- The os statement will execute a command to the os and can read the output of the commands to the os
- The commands that can be executed are those that can be run from:
  - Terminal window on OS X
  - Terminal window on Linux
  - Command Window on Windows
- The commands for each OS are different so the resulting code will be OS dependent
  - It is possible to fingerprint the OS and add code specific for each OS
- Example:
  - ifconfig in Linux and OS X gets the state of the network ports
  - ipconfig in Windows gets the state of the network ports

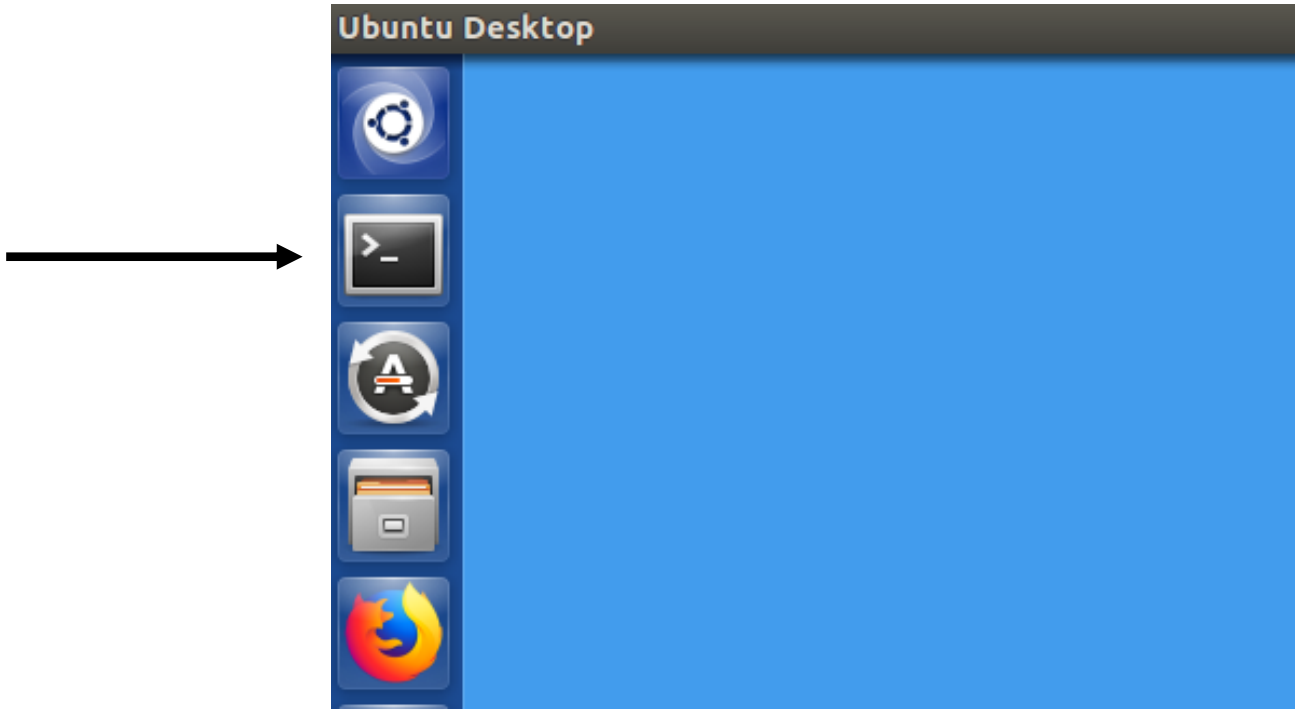
Terminal window in OS X



Command Prompt in Windows



Terminal Window in Linux



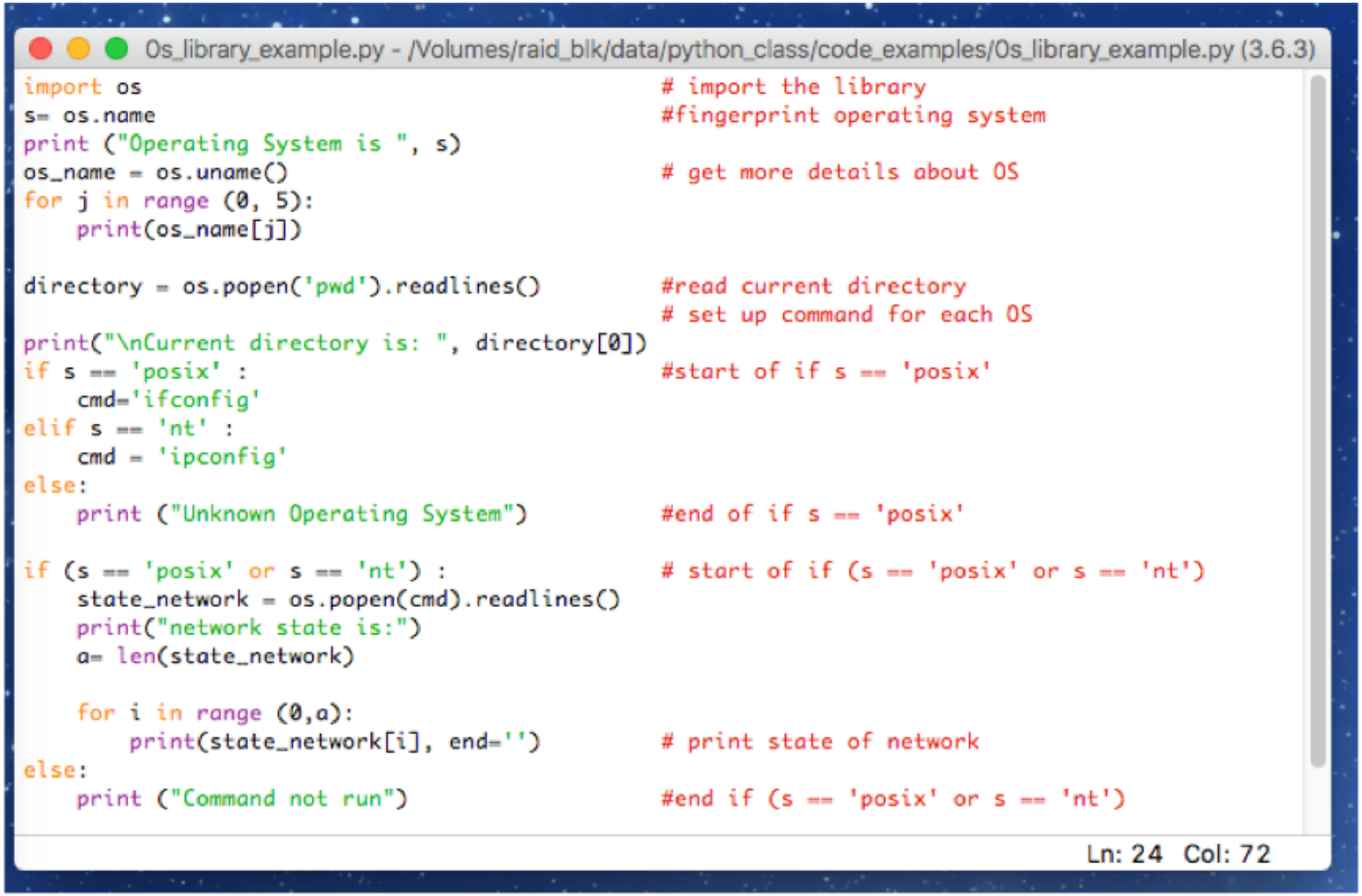
Some OS Calls

- os.system() Executing a shell command
- os.environ() Get the users environment
- os.getcwd() Returns the current working directory.
- os.getgid() Return the real group id of the current process.
- os.getuid() Return the current process's user id.
- os.getpid() Returns the real process ID of the current process.
- os.umask(mask) Set the current numeric umask and return the previous umask.
- os.uname() Return information identifying the current operating system.

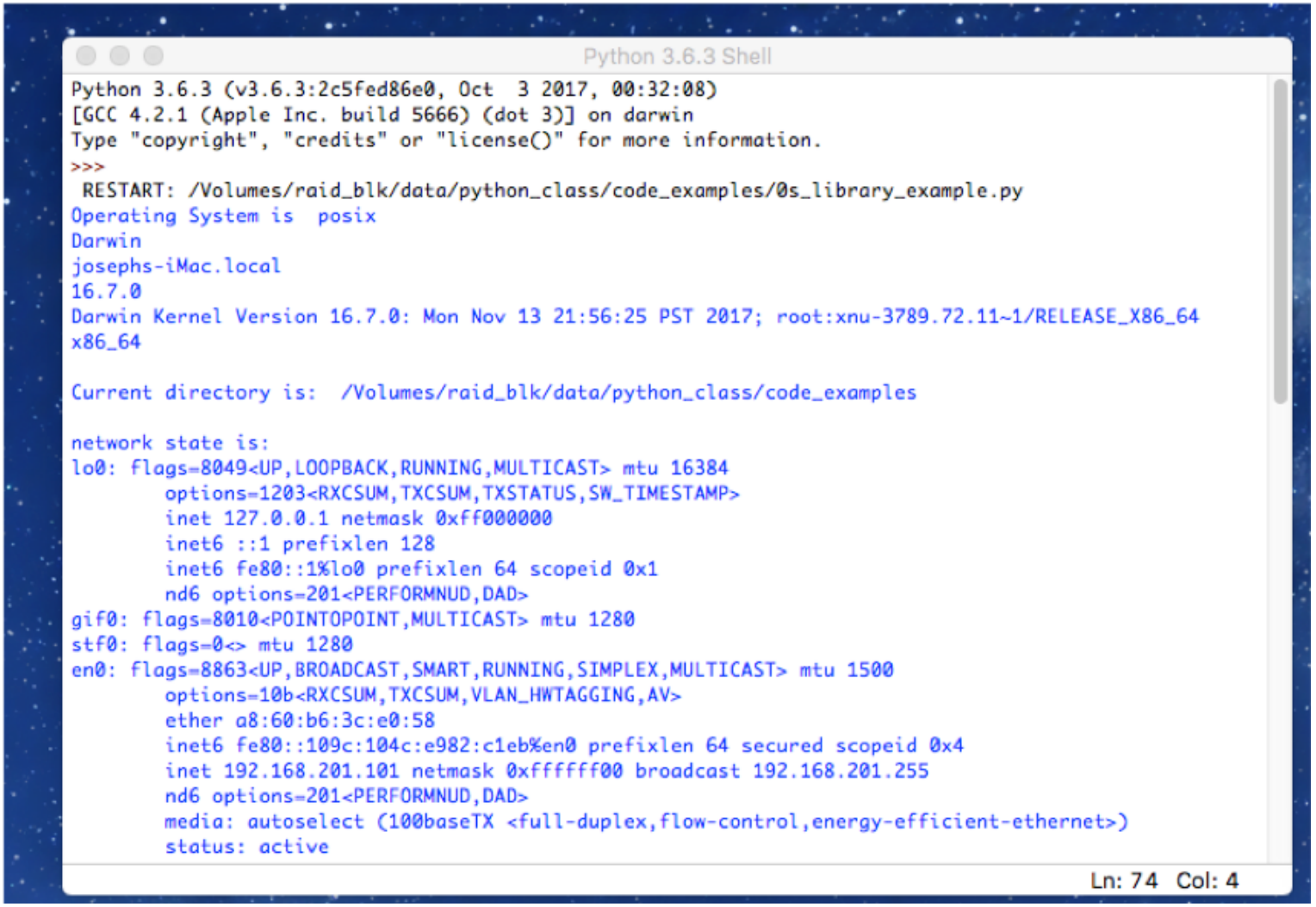
Some OS Calls (Con't)

- os.chroot(path) Change the root directory of the current process to path
- os.listdir(path) Return a list of the entries in the directory given by path.
- os.mkdir(path) Create a directory named path
- os.makedirs(path) Recursive directory creation function.
- os.remove(path) Remove (delete) the file path.
- os.removedirs(path) Remove directories recursively.
- os.rename(src, dst) Rename the file or directory src to dst.
- os.rmdir(path) Remove (delete) the directory path.
- os.popen('command').readlines() Execute 'command'; read the results

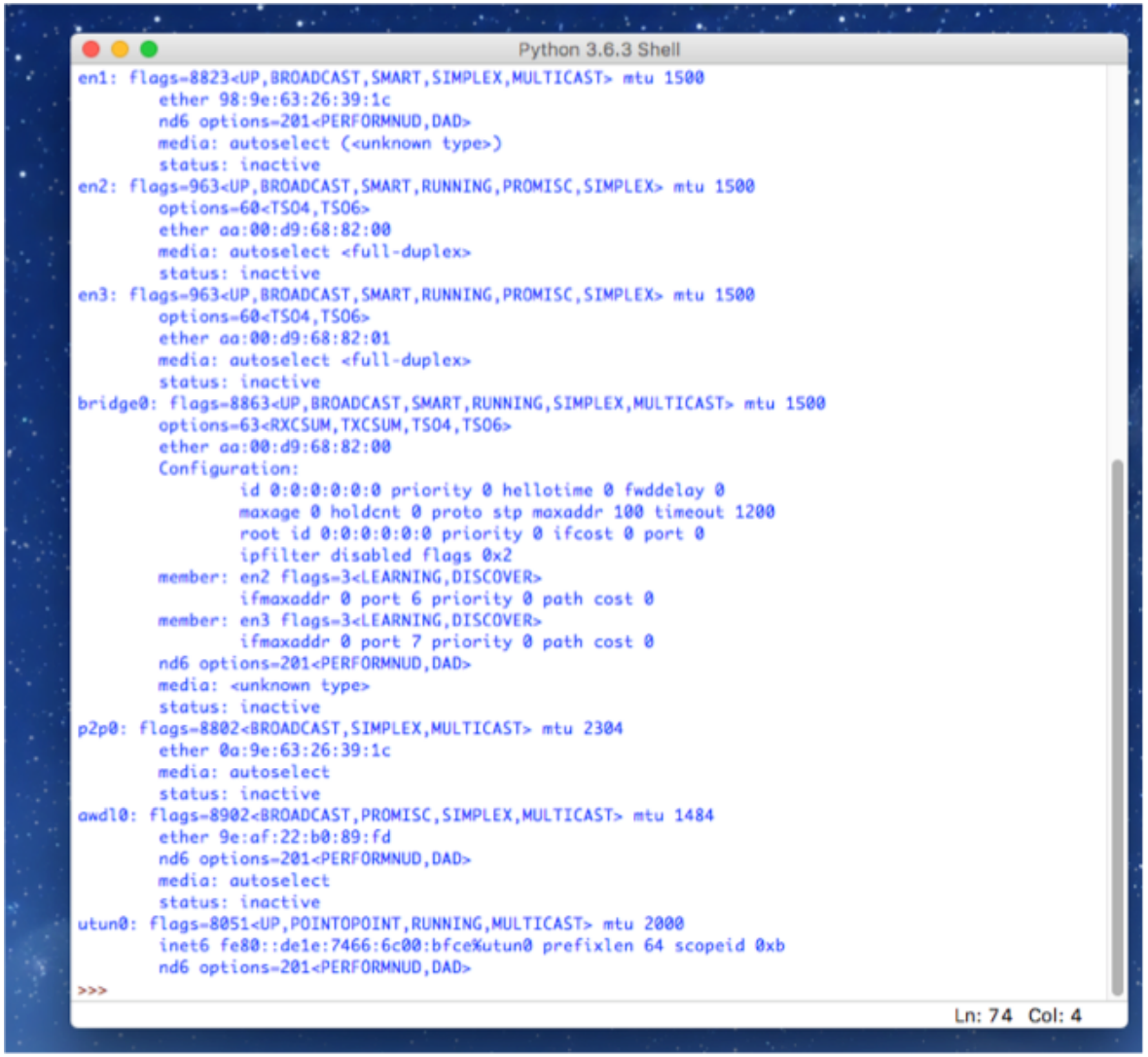
Example use of os library



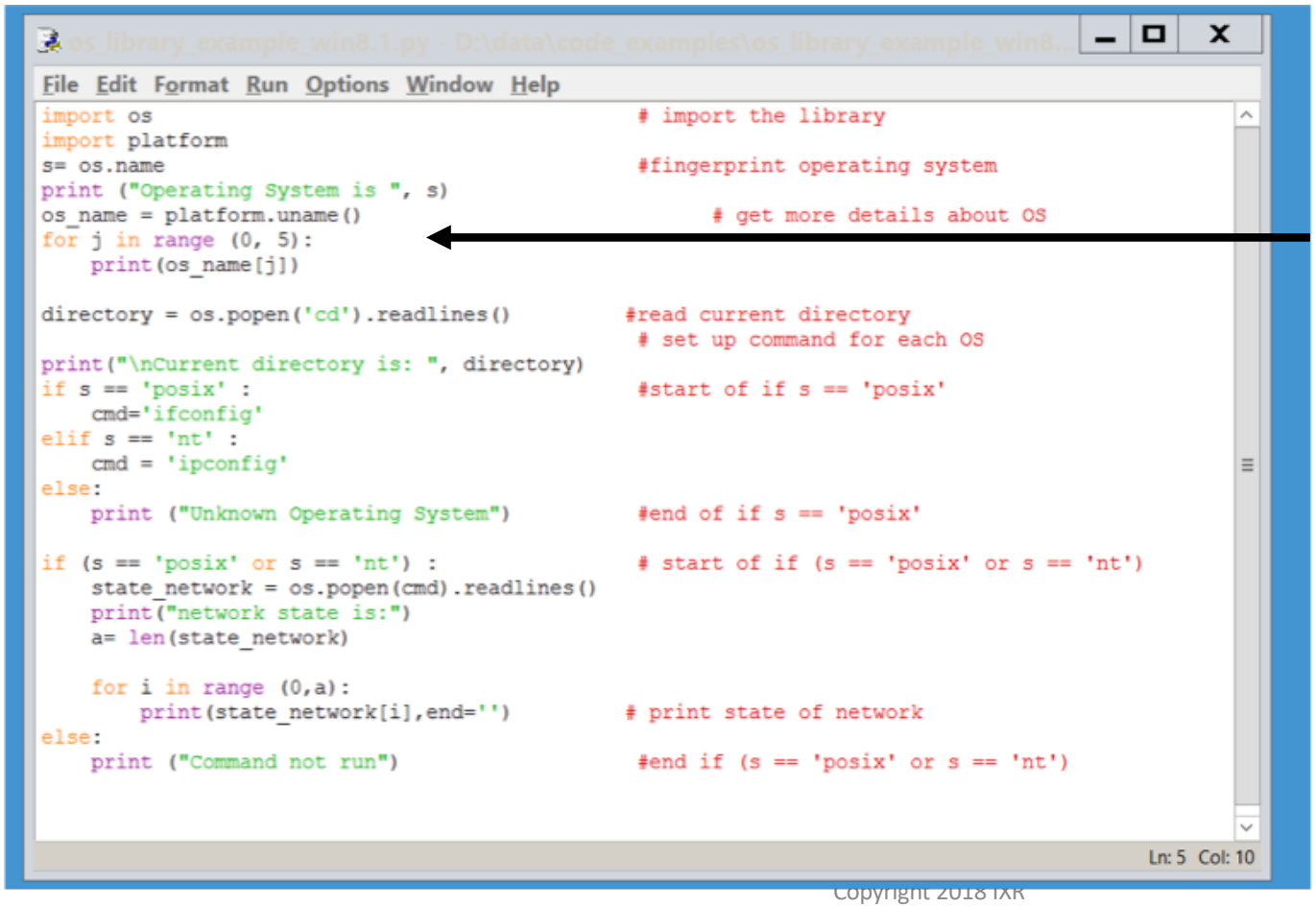
Output of example



Rest of Output

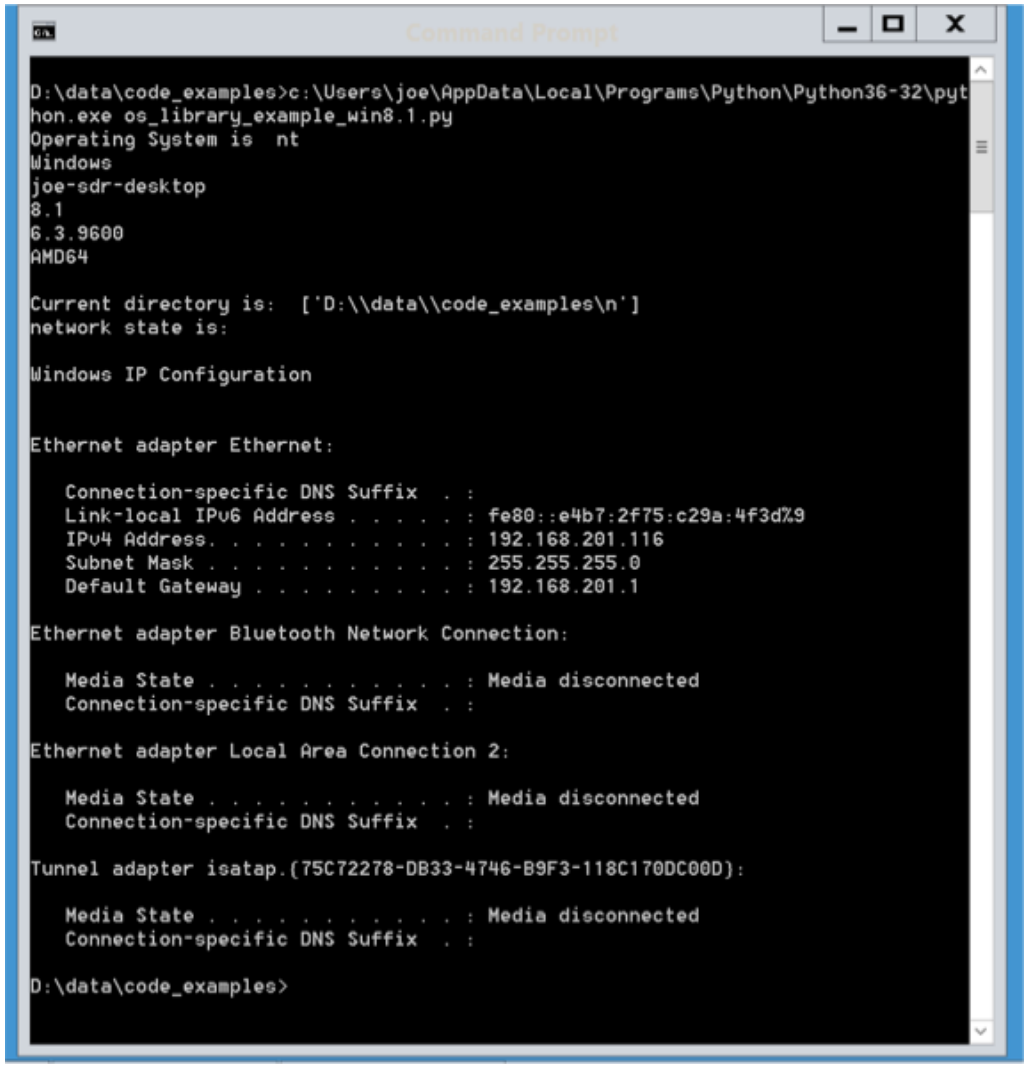


Examples of OS calls Windows

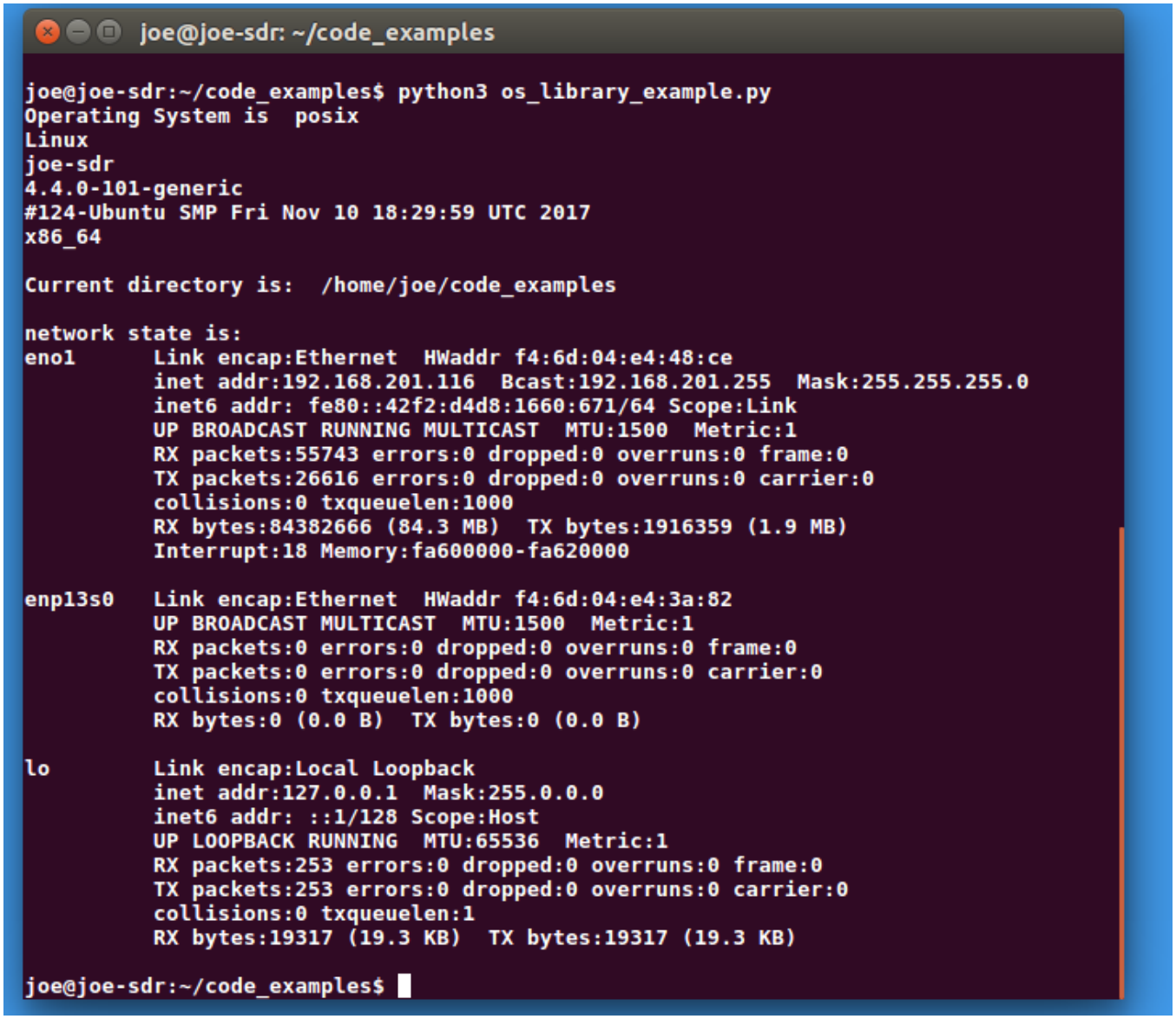


Command os.uname does not work in Windows Platform.uname() does work

Results



Example of OS calls in Linux



Home Work

- time.time() returns the time in seconds since January 1, 1970, 00:00:00 (UTC). When will this no longer work. i.e., at what year/date/hour will the clock reset to Jan 1, 1970
- The example pygame program goes off the screen with the circle is moved multiple times.
  - Change the program so when the circle goes off the screen it moves to the other side (up, down, left, right) like some old video games
  - This is called an unbounded universe
  - Bonus: Change the program so the direction of move changes when the circle hits the edge of the screen
    - This is called a bounded universe
  - Question of the ages: Is our real universe bounded or unbounded?
- Modify the os example to make one program for OS X/Linux/Windows