Python

Methodologies for Data Science

Lars Sorensen - 16:137:603
Fall 2015 - biglars@cs.rutgers.edu
Unit Three

Python Methodologies for Data Sciences

Take a breath. You've done a lot in the last month…

Loading Python on your machine - Figuring out how to run IDLE - Using the Python interactive command window
Creating programs - Print function - Variables - Assignment - Arithmetic
Modulus - Strings - Integers - Floats - Casting - Input function
Booleans - Arithmetic conditionals (>, <, ==, !=, <=, >=) - Boolean operators (And, Or, Not) - if - elif - else
Sequences - strings - slicing - indexing, reverse indexing - len function - ord, chr - string methods
Lists - split method - append method - mutability of lists - mixed data types of lists
Loops - Iteration - for loops - range function - nested loops - while loops - sentinel loops - counters - accumulators
Interactive loops - definite loops - indefinite loops - keyword 'in' as set conditional

Now that's a whole lot of stuff….

Python Methodologies for Data Sciences

Everyone is doing great. Unit 3 is the last of what I see as the "basics" of imperative (writing programs like recipes... remember) programming. I explained a bunch of what I'm talking about in the second in-person class, suffice it to say that I want the first half of the class to be absorbing the basics and the second half of the class to learn special programming topics with Data Science playing a role in the Unit assignments.

All that said, we're almost there. One more Unit to go. You know about data, you know about sequences. You now know how to navigate through your data with loops. Things are beginning to take shape. Just a few more major concepts to cover. They all relate to one topic though, modularity.

Modularity, compartmentalization, you pick the name. What we want to do is show that it is often more efficient to break down big problems into little ones and stack the little ones up until they solve the big ones. It makes code easier to understand and aides us in reusing pre-written code and saving time. OK. We're off to learn about functions, File I/O and using modules, or third party libraries.

Functions

Python Methodologies for Data Sciences

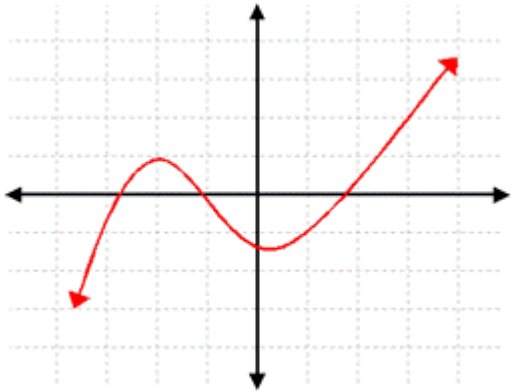
A function is a named sequence of statements that performs a calculation or computation. When we define a function we specify the name and the sequence of statements and then you "call" that function by name.

think about:

f(x) = x + 4

So… f(4) = 8 and f(6) = 10 and so on…

If we do g(x) = f(x) + 3 then g(3) = 10…



But with programming statements…

Python Methodologies for Data Sciences

Anatomy of a Function

The name of the function is ‘Type’

```
Python 3.3.0 (v3.3.0:bd8afb90ebf2,
D64)] on win32
Type "copyright", "credits" or "li
>>> type(64)
<class 'int'>
>>> type('Hello World')
<class 'str'>
>>>
```

The result is called the ‘return value’

The expression in parens is called the ‘argument’

A function can take an argument, processes it, and return a result. Some functions do not take an argument and some do not return a value.

Python Methodologies for Data Sciences

Some functions are built-in to Python and are ready for use. We have already seen a bunch of them.

The casting functions are all built-in functions in Python.

```
>>> int(32.4)
32
>>> int('78')
78
>>> float(34)
34.0
>>> str(34.56)
'34.56'
>>>
```

Later, when we see the math module, we will see that these functions, while not built-in to Python, can be imported from another file for us to use in our programs.

```
>>> import math
>>>
>>> math.sqrt(25)
5.0
>>>
>>> degrees = 45
>>> radians = degrees / 360.0 * 2 * math.pi
>>> math.sin(radians)
0.7071067811865475
>>>
```

Python Methodologies for Data Sciences

More functions that we have not used are built-in to Python and are ready for use as well.

You can find the documentation for these functions here:

https://docs.python.org/3/library/functions.html

Check out sum() and pow()

2. Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are list

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Python Methodologies for Data Sciences

Using built-in functions is convenient, but defining your own functions is easy. Use the 'def' keyword, give your function a name, a parameter list (none in the example, but the parenthesis are still required) and then a colon.

We then give the function a definition by coding what we want it to do. Here we just want it to print four lines of text.

Remember

- Functions have the same naming rules as variables.
- Functions must appear in the code before they are called.

funcs1.py - C:/SimplePython/PythonPrograms/funcs1.py

File Edit Format Run Options Windows Help

```
# this is a comment
#
# Author : Lars Sorensen
#
#  This program uses functions
#

def print_drc():
    print("I feel uptight on Saturday night...")
    print("Nine o'clock, the radio's the only light...")
    print("I hear my song and it pulls me through")
    print("Comes on strong, tells what I got to do...")


print("This is the function program\n")
print_drc()
```

Python Methodologies for Data Sciences

funcs1.py - C:/SimplePython/PythonPrograms/funcs1.py

File Edit Format Run Options Windows Help

```
# this is a comment
#
# Author : Lars Sorensen
#
#  This program uses functions
#

def print_drc():
    print("I feel uptight on Saturday night...")
    print("Nine o'clock, the radio's the only light...")
    print("I hear my song and it pulls me through")
    print("Comes on strong, tells what I got to do...")


print("This is the function program\n")
print_drc()
```

We see the code run below. We print the "This is..." line and then we call the function. Then the function named print_drc() prints four lines. Control returns to the spot after where the function was called and the program ends.

Interesting, but why bother?

```
>>> ================================ RESTART ================================
>>>
This is the function program

I feel uptight on Saturday night...
Nine o'clock, the radio's the only light...
I hear my song and it pulls me through
Comes on strong, tells what I got to do...
>>>
```

Python Methodologies for Data Sciences

funcs2.py - C:/SimplePython/PythonPrograms/funcs2.py

File Edit Format Run Options Windows Help

```
# this is a comment
#
# Author : Lars Sorensen
#
#  This program uses functions
#

def print_drc():
    print("I feel uptight on Saturday night...")
    print("Nine o'clock, the radio's the only light...")
    print("I hear my song and it pulls me through")
    print("Comes on strong, tells what I got to do...")

def print_drc_twice():
    print_drc()
    print_drc()

print("This is the function program\n")
print_drc_twice()
```

Because it cuts down on repetition. I can run the function twice and print 8 lines.

What if I wanted to print it 10 times?

```
>>> ================================ RESTART ================
>>>
This is the function program

I feel uptight on Saturday night...
Nine o'clock, the radio's the only light...
I hear my song and it pulls me through
Comes on strong, tells what I got to do...
I feel uptight on Saturday night...
Nine o'clock, the radio's the only light...
I hear my song and it pulls me through
Comes on strong, tells what I got to do...
>>>
```

Python Methodologies for Data Sciences

```python
# this is a comment
#
# A: Lars
#
# Playing w Functions

def print_drc():
    print("\tI feel uptight on Saturday night...")
    print("Nine o'clock, the radio's the only light...")
    print("\tI hear my song and it pulls me through")
    print("Comes on strong, tells what I got to do...")

for i in range(10):
    print_drc()
```

```
Nine o'clock, the radio's the only light...
        I hear my song and it pulls me through
Comes on strong, tells what I got to do...
        I feel uptight on Saturday night...
Nine o'clock, the radio's the only light...
        I hear my song and it pulls me through
Comes on strong, tells what I got to do...
        I feel uptight on Saturday night...
Nine o'clock, the radio's the only light...
        I hear my song and it pulls me through
Comes on strong, tells what I got to do...
        I feel uptight on Saturday night...
Nine o'clock, the radio's the only light...
        I hear my song and it pulls me through
Comes on strong, tells what I got to do...
        I feel uptight on Saturday night...
Nine o'clock, the radio's the only light...
        I hear my song and it pulls me through
Comes on strong, tells what I got to do...
        I feel uptight on Saturday night...
Nine o'clock, the radio's the only light...
        I hear my song and it pulls me through
Comes on strong, tells what I got to do...
        I feel uptight on Saturday night...
Nine o'clock, the radio's the only light...
        I hear my song and it pulls me through
Comes on strong, tells what I got to do...
```

Now we can see that using functions is saving us time. Without the use of functions it would take 40 lines of code to do this program. With functions we can cut that down to seven. All we did was to run the function inside of a loop so it was run 10 times.

Python Methodologies for Data Sciences

```
# this is a comment
#
# A: Lars
#
# Playing w Functions

def print_drc():
    print("\tI feel uptight on Saturday night...")
    print("Nine o'clock, the radio's the only light...")
    print("\tI hear my song and it pulls me through")
    print("Comes on strong, tells what I got to do...")

for i in range(10):
    print_drc()
```

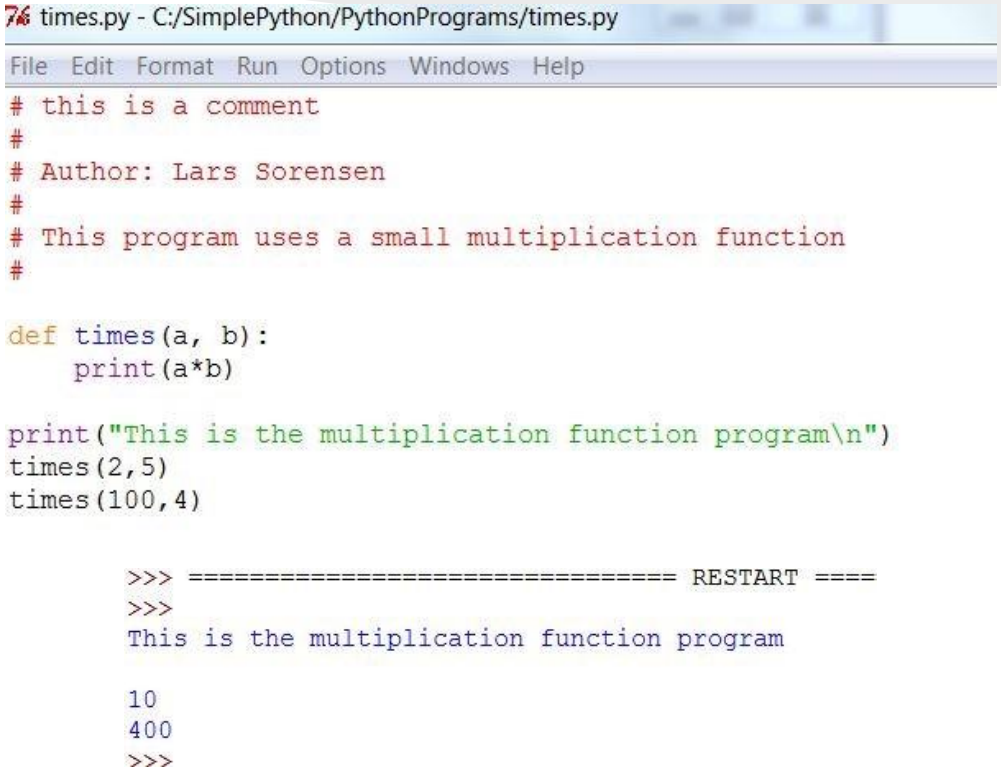
This is a very simple function. It takes no parameters and it returns no values.

We can save time and coding this way, but we can do much more by passing parameters and returning values.

Python Methodologies for Data Sciences

Here we define a function that takes two numbers as arguments. Our function takes the two numbers and prints their product.

As we have seen, this function requires 'arguments.' At the call of the function those variable names are called parameters. In the function definition's header (the line with 'def' in it), we call them arguments. An argument can be a variable or an expression. Inside the function they act as normal variables.



Python Methodologies for Data Sciences

times.py - C:\SimplePython\PythonPrograms\times.py

File Edit Format Run Options Windows Help

```
# this is a comment
#
# Author: Lars Sorensen
#
# This program uses a small multiplication function
#

def times(a, b):
    print(a*b)

print("This is the multiplication function program\n")
times(2,5)
times(100,4)
print(a)
```

```
>>> ================================ RESTART ================================
>>>
This is the multiplication function program

10
400
Traceback (most recent call last):
  File "C:\SimplePython\PythonPrograms\times.py", line 14, in <module>
    print(a)
NameError: name 'a' is not defined
>>>
```

Scope

Look at the last line of our program. We ask to print the variable a, but if you look at the run we see that this produces an error.

When the function ends, so does the existence of the variables within it. They are called “local” variables and cannot be accessed outside of their “scope” or the function where they are defined and used.

Python Methodologies for Data Sciences

Return Values

Now we've put it all together. We have a definition with parameters and a function body that returns a value. Now that's a function!

Once you return a value the function call is over. You can return any variable type you like int, float, string, or even lists.

Any code after a return is reached will not be executed and is referred to as 'dead code.' Creepy.

In my comment in the code to the right I said this was a "fruitful function." What's that all about?

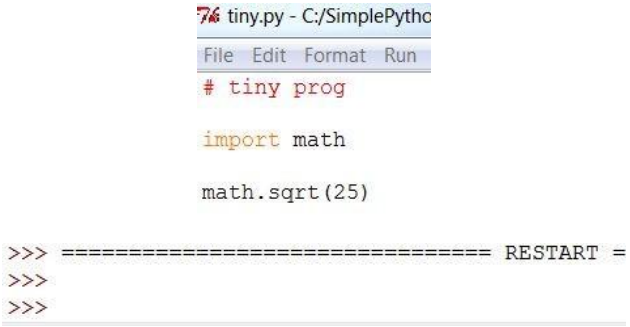
File Edit Format Run Options Windows Help

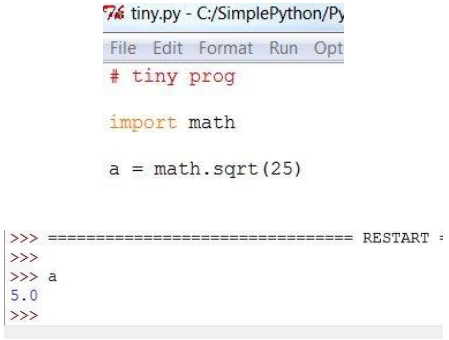
```
# this is a comment
#
# Author: Lars Sorensen
#
# This program uses a small multiplication function
#

def times(a, b):
    return a*b

print("This is the multiplication function program\n")
a = times(2,5)  # remember, times is a fruitful function
print(a)
```

```
>>> ================================ RESTART ==
>>>
This is the multiplication function program

10
>>>
```

Python Methodologies for Data Sciences

Fruitful vs Void





Some functions return values, like math.sqrt(25) *(We will discuss importing modules in five minutes ;-), well, actually, after a break...)* and some just perform a group of actions, like the function we wrote to print the song lyrics. The issue here is that if you use a fruitful function, one that returns a value, in a program and do not assign the result to a variable it will be lost forever!

Python Methodologies for Data Sciences

I wrote up this snippet to illustrate some of the behaviors of functions.

```
#
# This is a comment
#
# A: Lars
#
# Playing around w Functions

before = 'GLOBAL'

def times(a, b):
    # just a comment
    print("this is the times function")
    print("Inside IDs",id(num1),id(num2))
    print("Can I print a global var decl bf the func def?", before)
    print("Can I print a global after the func def?", after )
    a = a+1
    print(id(a))
    mylist[2] = 7
    return a*b

after = "CALL"
mylist = [2,3,4,5]
num1, num2 = 3,12
print("Outside IDs",id(num1),id(num2))
print(times(12,3))
print(mylist)
print(num1)
```

Python Methodologies for Data Sciences

Can you trace through this code? It teaches us a few things about scope, returning values from fruitful functions and a few other things. I will be tracing this code in the first review video, but see if you can do it yourself first.

```
>>>
Outside IDs 2012395248 2012395536
this is the times function
Inside Function IDs 2012395248 2012395536
Can I print a global var decl bf the func def? GLOBAL
Can I print a global after the func def? CALL
2012395568
39
[2, 3, 7, 5]
3
>>>
```

```
#
# This is a comment
#
# A: Lars
#
# Playing around w Functions

before = 'GLOBAL'

def times(a, b):
    # just a comment
    print("this is the times function")
    print("Inside IDs",id(num1),id(num2))
    print("Can I print a global var decl bf the func def?", before)
    print("Can I print a global after the func def?", after )
    a = a+1
    print(id(a))
    mylist[2] = 7
    return a*b

after = "CALL"
mylist = [2,3,4,5]
num1, num2 = 3,12
print("Outside IDs",id(num1),id(num2))
print(times(12,3))
print(mylist)
print(num1)
```

Functions

So we have different types of functions….

Void – these functions do not return a value
Fruitful – these functions do return a value
Functions with no arguments – just names for blocks of code
Functions with arguments – parameters that are used inside the function, like our times function

Why Bother with Functions?

Creating functions gives us a way to name a block of code. This can make programs more readable, and easy to debug.

Functions make programs smaller because you do not have to retype things over and over again. Also, if you want to make a change later you only have to do so in one place.

Dividing programs into smaller hunks make them easier to manage when your programs begin to get larger. This ‘modularity’ will be discussed in the second half of this Unit.

Well designed functions can be used by more than one program! You can save time and energy by merely reusing your functions later!

Python Methodologies for Data Sciences

Ok, Now you have functions under your belt. take a break. Think about it. Now you can have a bunch of sub-programs and one main program to call them all. It's all so organized.

When I do my Unit 3 review video I will not only trace the crazy function I listed before, but I will re-write Euler 6 using functions to make solving this problem even easier AND to create some reusable code along the way.

Take a break, you deserve it, but come back for the second half of the Unit where we look at File I/O (that's Input and Output) and we learn about modularity and how we can keep our code in multiple files and begin to use other pre-written libraries for our programs.

```
print("\nThe Euler 6 Program\n")
limit = int(input("Enter the last number of the range: "))

# Find the sum of the squares
sos = 0
for i in range(1,limit+1):
    sos = sos + i**2

print("sum of squares is", sos)

# Find the square of the sum
thesum = 0
for i in range(1,limit+1):
    thesum = thesum + i

sosq = thesum**2
print("The square of the sum is",sosq)

# find the answer
answer = sosq - sos
print("The answer is ",answer)
```

Python Methodologies for Data Sciences

We're almost there. After we finish this unit you will have the basics of Python down....

But for now take a break, you need to absorb all of that information about functions...

Ready? OK, good. Go to the next slide for the home stretch where we learn about files and then modularity...

Files

Python Methodologies for Data Sciences

Python Methodologies for Data Sciences

The picture to the right is a screenshot of a data file with 5,163 names in it. That may seem like a lot, but this is actually a small and manageable data set.

Files are resources for data that are based on “durable” storage. That’s a fancy way of saying that most of our data so far has been held in RAM (random access memory) and when we turn the computer off it goes away. Files are usually written to a hard disk drive (or to a hard disk drive somewhere else that we access across the Internet. Years ago this was called online data and it scared people, they thought this was insecure and dangerous. But now, ten years later, with the only difference being that we call it the “cloud”, the less intelligent among us can sleep at night and can’t seem to get enough Dropbox or Drive storage for their mortgage spreadsheets and cat pictures. I have no idea why this is.)

Durable file storage can save data, allow the computer to be turned off and then still be accessible when the machine is turned back on.

Python Methodologies for Data Sciences

Reading from files in Python is easy. It's a three step operation. First we associate our file with a variable name. In the old days this used to be called a "handle." We then use this "file handle" or variable to read in the data.

When we are done using the data we can close our file and let Python know that we are done with it and will not refer to it again.

```
# This is a comment
#
# Author : Lars Sorensen
#
# This program just reads from a file and prints it...
#

print("\nThis is the file reading program\n")

fname = input("Enter a filename: ")
infile = open(fname,"r")
data = infile.read()
print(data)

infile.close()
```

Python Methodologies for Data Sciences

Open

Read

Close

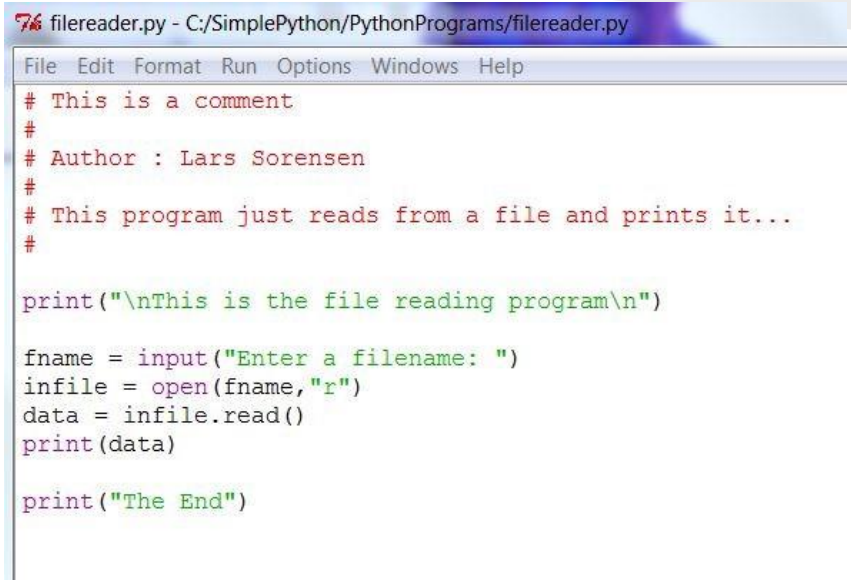
```
# This is a comment
#
# Author : Lars Sorensen
#
# This program just reads from a file and prints it...
#

print("\nThis is the file reading program\n")

fname = input("Enter a filename: ")
infile = open(fname,"r")
data = infile.read()
print(data)

infile.close()
```

Python Methodologies for Data Sciences

In the SAKAI resources folder Unit three I have a file called ‘numbers.dat’

Code up the program to the right and use it on numbers.dat. Remember, you have to keep your program and your .dat file in the same directory.

What happens?



Python Methodologies for Data Sciences

filereader.py - C:/SimplePython/PythonPrograms/filereader.py

File Edit Format Run Options Windows Help

```
# This is a comment
#
# Author : Lars Sorensen
#
# This program just reads from a file and prints it...
#

print("\nThis is the file reading program\n")

fname = input("Enter a filename: ")
infile = open(fname,"r")
data = infile.read()
print(data)

print("The End")
```

```
This is the file reading program

Enter a filename: numbers.dat
This is my number file
12
23
34
45
33
66
76
53
67

The End
>>>
```

You should see something just like this. Python prints the file out just as it found it. Email me and tell me how many characters are in this file and I will give you an extra credit point towards the quiz for unit three.

Python Methodologies for Data Sciences

```
# This is a comment
#
# Author : Lars Sorensen
#
# This program just reads from a file and prints it...
#

print("\nThis is the file reading program\n")

fname = input("Enter a filename: ")
infile = open(fname,"r")
data = infile.read()
print(data)

print(data[2], data[22])
```

```
This is the file reading program

Enter a filename: plaintext.txt
All children, except one, grow up. They soon know that they will grow up, and the way
Wendy knew was this. One day when she was two years old she was playing in a garden, a
nd she plucked another flower and ran with it to her mother. I suppose she must have l
ooked rather delightful, for Mrs Darling put her hand to her heart and cried, 'Oh, why
can't you remain like this for ever!' This was all that passed between them on the sub
ject, but henceforth Wendy knew that she must grow up. You always know after you are t
wo. Two is the beginning of the end.
l n
```

The last line of the program shows us that data is just a string. It's a big string though. It contains the entire first paragraph of the book Peter Pan.

At the bottom of the program we print the 3rd and 23rd characters from the string. Count into the first sentence and you'll see...

Python Methodologies for Data Sciences

```
namesFile = open("E22.txt","r")
names = namesFile.read()
# This gets rid of the double quotes
names = names.replace('"', "")
# This creates a list fm the string w the comma as the delimeter
names = names.split(",")
# God bless sort methods...
names.sort()
```

Do you remember when I loaded all those names into a list last Unit so we could look for my name? I told you that you would have to wait to see how I loaded them from a file? Well, the wait is over...

Python Methodologies for Data Sciences

Here is a program that reads the file (we do not ask the user any longer, we hardcode the name of our file in this example) and manipulates it so we can put the names from the file into a list.

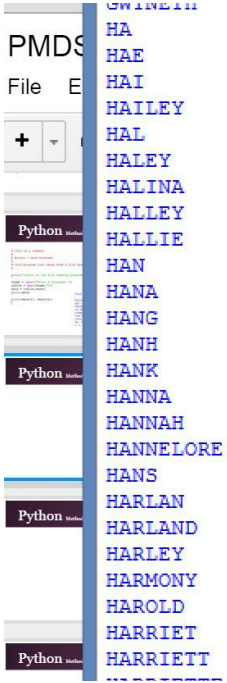
E22.txt is our file with 5,162 names in it. I read it into a string called names. I use replace to get rid of the quotes (we will do this to clean up data files often), and we use split to take the parts of the string delimited (just a place where we know to break, here when we see a comma) by a comma (the delimiter, the comma, is not saved) and create a list with it.

Once we have a list we can iterate over it, print it, do whatever we like with it.

```
# Author Lars
#
# Load the names file into a list

# Open the names file and create a big string called names with the
# whole file in it

namesFile = open("E22.txt","r")
names = namesFile.read()

# print the length
print("This file has", len(names), "characters in it.\n")

# Lets make this a list
# This gets rid of the double quotes
# it takes a " and makes it blank or ""
names = names.replace('"', "")
# This creates a list fm the string w the comma as the delimeter
names = names.split(",")

# now I have a list with my names as items.
# God bless sort methods...
names.sort()

# print the items out - wait for input tho
wait = input("Press enter to list names")
for i in names:
    print(i)
```

Python Methodologies for Data Sciences

```
# Author Lars
#
# Load the names file into a list

# Open the names file and create a big string called names with the
# whole file in it

namesFile = open("E22.txt","r")
names = namesFile.read()

# print the length
print("This file has", len(names), "characters in it.\n")

# Lets make this a list
# This gets rid of the double quotes
# it takes a " and makes it blank or ""
names = names.replace('"', "")
# This creates a list fm the string w the comma as the delimeter
names = names.split(",")

# now I have a list with my names as items.
# God bless sort methods...
names.sort()

# print the items out - wait for input tho
wait = input("Press enter to list names")
for i in names:
    print(i)
```



This program should print all 5,168 names to the screen (it takes a **while**, press CTRL-C if you're bored with it.) This is not how long it takes to iterate through the file (we'll actually show this with the time module later), but how long it takes to display things on the screen.

Note what we did with the input statement to create a pause...

Python Methodologies for Data Sciences

Because this is Python there has to be some special, easy “Pythonic” way of doing things efficiently.

Python gives us a special way to iterate through a file line by line. By using “with” and “as” we can handle opening and closing automatically and iterate through a file line by line.

Look at what the program to the right does very carefully.

```
# This is a comment
#
# Author : Lars Sorensen
#
# This program reads in Numbers.dat and finds the
# sum of the numbers, skipping the header
#

accum = 0
with open("numbers.dat", "r") as f:
    for line in f:
        if (line[0:4] == 'This'):
            pass
        else:
            accum = accum + int(line)

print("The sum of the file's numbers is:", accum)
```

Python Methodologies for Data Sciences

```
# This is a comment
#
# Author : Lars Sorensen
#
# This program reads in Numbers.dat and finds the
# sum of the numbers, skipping the header
#

accum = 0
with open("numbers.dat", "r") as f:
    for line in f:
        if (line[0:4] == 'This'):
            pass
        else:
            accum = accum + int(line)

print("The sum of the file's numbers is:", accum)
```

This is the output of the program. Notice how the first thing we do in the loop is to test for the header. When we see it we want to do nothing. We can do this is Python with the **pass** keyword.

Then, we iterate through the file, casting the string numbers to integers and adding them to an accumulator. When the loop is done we print the sum.

```
>>>
The sum of the file's numbers is: 409
>>>
```

When in doubt, use open and close, but if you know that you want to quickly open a file and run through it line by line, using with-as is a good technique.

Python Methodologies for Data Sciences

Writing to files

We can read from files to get data to use in our programs, but we can also create new files with the data we create with our programs.

To write to a file we use the “open” function to attach the file to a variable, but this time we use a “w” to let Python know we intend to write data to this file. Then, using the out file variable, we use the write method to send the data to the writable file.

Examine the code to the right. Can you tell what it does?

```
# This is a comment
#
# Author : Lars Sorensen
#
# This program reads from a file and writes it to
# another file.... backwards!
#

print("\nThis is the backwards output program\n")

rname = "plaintext.txt"
oname = "backwards.txt"

rFile = open(rname, "r")
oFile = open(oname, "w")

data = rFile.read()
print(len(data))

for i in range(len(data)-1, -1, -1):
    oFile.write(data[i])

rFile.close()
oFile.close()
```

Python Methodologies for Data Sciences

```
# This is a comment
#
# Author : Lars Sorensen
#
# This program reads from a file and writes it to
# another file.... backwards!
#

print("\nThis is the backwards output program\n")

rname = "plaintext.txt"
oname = "backwards.txt"

rFile = open(rname, "r")
oFile = open(oname, "w")

data = rFile.read()
print(len(data))

for i in range(len(data)-1, -1, -1):
    oFile.write(data[i])

rFile.close()
oFile.close()
```

I used my file reading program below to examine my new file, backwards.txt. The program read the entire first chapter of Peter Pan into the string data. Then, character by character, the program writes it to a new file, backwards.

Examine the for loop closely to see what we did.

```
This is the file reading program

Enter a filename: backwards.txt
.dne eht fo gninnigeb eht si owT .owt era uoy retfa wonk syawla uoY .pu worg tsum ehs
taht wenk ydneW htrofecneh tub ,tcejbus eht no meht neewteb dessap taht lla saw sihT '
!reve rof siht ekil niamer uoy t'nac yhw ,hO' ,deirc dna traeh reh ot dnah reh tup gni
lraD srM rof ,lufthgiled rehtar dekool evah tsum ehs esoppus I .rehtom reh ot ti htiw
nar dna rewolf rehtona dekculp ehs dna ,nedrag a ni gniyalp saw ehs dlo sraey owt saw
ehs nehw yad enO .siht saw wenk ydneW yaw eht dna ,pu worg lliw yeht taht wonk noos ye
hT .pu worg ,eno tpecxe ,nerdlihc llA
>>>
```

Python Methodologies for Data Sciences

```
# This is a comment
#
# Author : Lars Sorensen
#
# This program reads from a file and writes it to
# another file.... backwards!
#

print("\nThis is the backwards output program\n")

rname = "backwards.txt"
oname = "plainredux.txt"

rFile = open(rname, "r")
oFile = open(oname, "w")

data = rFile.read()
print(len(data))

for i in range(len(data)-1, -1, -1):
    oFile.write(data[i])

rFile.close()
oFile.close()
```

Look at the file names in the code to the left. I just took the file that was output in the previous program and I'm reading it in as input for this program. To test things I reversed my output file yet again to return to the original text. As you can read below, it worked.

The output files you write to one moment become the input files for other jobs later on. We can also open an existing file and "append" to it, but that is for another time...

```
This is the file reading program

Enter a filename: plainredux.txt
All children, except one, grow up. They soon know that they will grow up, and the way
Wendy knew was this. One day when she was two years old she was playing in a garden, a
nd she plucked another flower and ran with it to her mother. I suppose she must have l
ooked rather delightful, for Mrs Darling put her hand to her heart and cried, 'Oh, why
can't you remain like this for ever!' This was all that passed between them on the sub
ject, but henceforth Wendy knew that she must grow up. You always know after you are t
wo. Two is the beginning of the end.
>>>
```

Python Methodologies for Data Sciences

Pickles?

Python Methodologies for Data Sciences

As usual, Python has a quick and nifty way to write your data and read it back in a more Pythonic fashion. It's called Pickling.

When we get more involved with Python data types we will see that there are a myriad of ways that we can structure our data. Using lists to organize the elemental types like Integer, Float and Strings is just scratching the surface. We will soon see objects that can keep many different kinds of data.

In the past, saving these objects was a problem. A file wants a sequential list of characters or binary digits. Programmers would have to unpack their objects and make them linear in order to save them. This process was called "serialization." It was a pain. Now, we don't have to go through that tedious process any longer, most major languages provide ways to write objects to data files (see JSON for another popular method). Python lets us "pickle" them.

Python Methodologies for Data Sciences

In the code to the left we pickle a list.

First we type ‘import pickle’. This allows us to use the pickle code in the built in pickle library. Can you say, “Great segue Lars”? We’ll be learning about modules and libraries in about 5 minutes. More soon...

First we create a list. Then we open a file for writing. The ‘b’ next to the ‘w’ means that we want to open the file in ‘binary’ mode, we’ll be saving our data in binary.

Then we use the pickle library’s “dump” (an old and semi unfortunate computer science term for quickly offloading and saving data to file or device) method. This saves our list to a file.

This is not very helpful if we cannot read the list back when we need it. We test this by opening the file again, only this time for reading binary data. We then use the pickle.load method to get our object (in this case a list) back from the file. When we print it we see that it is our list.

```
# This is a comment
#
# Author Lars
#
# Playing w pickle

import pickle

# create a list
myList = [ 3,2,4,5,6,7,8,54,2,4,56,12,'four',23,23.4,56,22.4,55,'six']
print("My list before Pickling")
print(myList)

# save my list to a pickle file
oFile = "picklefile.dat"
# open the file for writing
myFile = open(oFile,'wb')

# this writes the object a to the
# file named 'testfile'
pickle.dump(myList,myFile)
# here we close the fileObject
myFile.close()

# We have saved the List and maintained the structure
# Now we test
# we open the file for reading
fileObject = open(oFile,'rb')
# load the object from the file into var b
b = pickle.load(fileObject)
print("Back from pickling:")
print(b)
```

Python Methodologies for Data Sciences

Don't type this code in. I saved it to a file called "pickle.py" and put it up in the SAKAI resources for Unit 3.

Go there, download it, and run it for yourself.

Now change the List and try it again.

Now try to save Integers and Floats. What happens?

```
# This is a comment
#
# Author Lars
#
# Playing w pickle

import pickle

# create a list
myList = [ 3,2,4,5,6,7,8,54,2,4,56,12,'four',23,23.4,56,22.4,55,'six']
print("My list before Pickling")
print(myList)

# save my list to a pickle file
oFile = "picklefile.dat"
# open the file for writing
myFile = open(oFile,'wb')

# this writes the object a to the
# file named 'testfile'
pickle.dump(myList,myFile)
# here we close the fileObject
myFile.close()

# We have saved the List and maintained the structure
# Now we test
# we open the file for reading
fileObject = open(oFile,'rb')
# load the object from the file into var b
b = pickle.load(fileObject)
print("Back from pickling:")
print(b)
```

Python Methodologies for Data Sciences

Behind the Curtain

The Data Science Tie-in

The next time you read or hear about “Marshalling Big Data.” you can think about pickling.

Marshalling is what us computer geeks did to save objects before we started calling it serialization in the late eighties and nineties.

Now, for some unknown reason, we are using the old Marshalling routines for Data Warehousing and Data Mining again. This is likely because, while older, Marshalling is seen as a more complete process as data AND an object’s codebase are maintained whereas with a regular serialized object we only rebuild the data component.

Python Methodologies for Data Sciences

Two thirds of the way through.

Take a break. Think about what we have covered with functions and file I/O.

After that continue on to find out about modularity and how we can not only use code that's been pre-written and kept in a library for us, but see how we can break up our code into different files to keep it more organized.

Modularity

Python Methodologies for Data Sciences

Modularity is the idea that when we start solving more complex problems with our programs we are going to want to start breaking them down into smaller sub-problems, creating code for these smaller problems and then stringing together our small solutions to provide a big solution for the complex problem.

That sounds complicated, but it's not. What we are going to discover is that there is a world of pre-written code out there for us to use and that it is simple to begin creating and saving our own code, getting more organized and promoting code reuse.

Python Methodologies for Data Sciences

Remember at the beginning of the unit we discussed the built in functions?

Imagine if there were scores and scores more functions for us to use in our programming. It turns out that there is, but if Python were to load all of them into memory it would become very sluggish and slow.

To combat this we have to tell Python that we want to “load” or “import” particular libraries so we can use the functions they offer. In this way we only load into memory what we are going to use.

2. Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are list

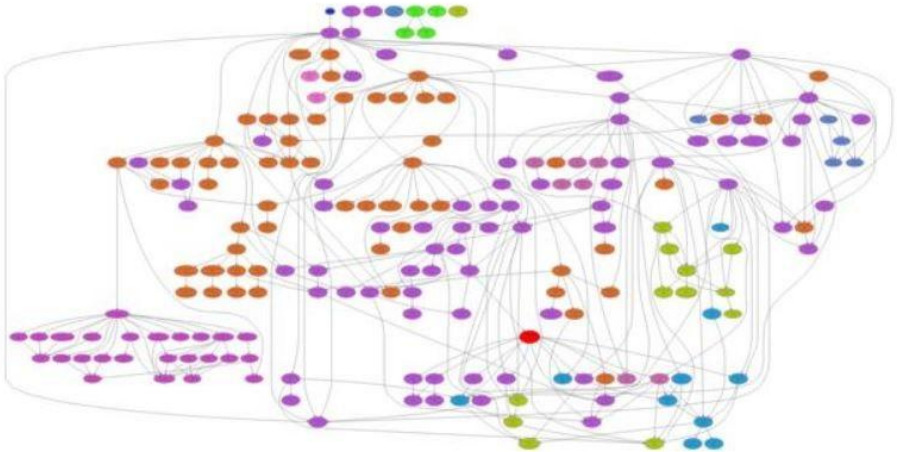
		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Python Methodologies for Data Sciences

Importing and Using Python Standard Library Modules

Python offers a rich library of pre-written code for us to use in our programs.

Clicking on the link below will take you to the documentation for the Python Standard Library. Being in the standard library just means that these functions (and classes, we'll get to that in unit 5) are always available to Python programmers when they install Python on their systems.



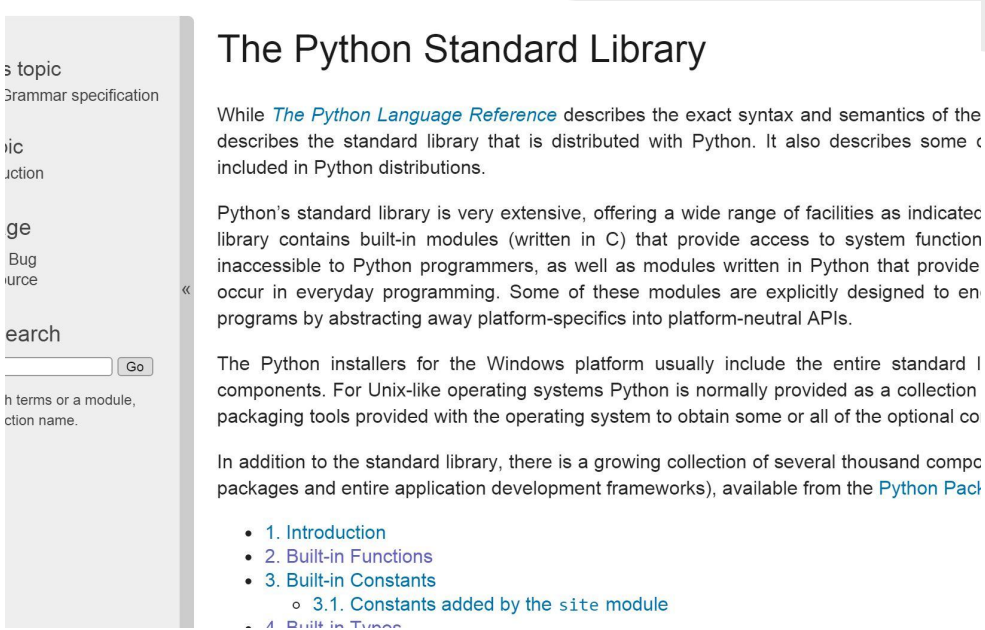
https://docs.python.org/3/library/

Python Methodologies for Data Sciences

Standard Library

The standard library documentation lets you know what is available to you. It lists the different "modules" that you can import into your programs and use in your code.

We are going to look at the Python import command and use it to import functions from three different modules from the Standard library: math, statistics and time.



https://docs.python.org/3/library/index.html

Python Methodologies for Data Sciences

Import

Look to the right. We have a module in the standard library called "math." With the import command we tell Python that we are going to be using functions from this library.

As you can see, we use the math code by calling a function with the name of the module (math), a period, and then the function we want to use.

We use the pre-written "sqrt" function to get square roots, we can translate degrees to radians (Pythons trig functions use radians) and we can even use a 'constant.' (pi). A constant is just a variable whose value never changes. The last time I looked pi was still 3.141592... so this is a good candidate for a constant variable. The math module knows this and defines it ahead of time for us.

```
# This is a comment
#
# Author: Lars
#
# Play with modules

import math

a = 34
b = -12
c = 34.12

print(math.sqrt(34))
print(math.pi)
print(abs(b))
print(math.factorial(6))
print(math.sin(math.radians(90)))
print(math.ceil(c))
```

```
>>>
5.830951894845301
3.141592653589793
12
720
1.0
35
>>>
```

Python Methodologies for Data Sciences

We can also use the 'from' command to tell Python to load particular functions and constants into our 'namespace.' This is just a fancy way of saying that we don't have to put 'math.' in front of our calls, we have tied this to the math module with our from command.

As you can see to the right, now we just have access to sqrt, pi or ceil (ceiling, it just rounds up) and do not have to call them with math.* in front of the function call.

We can also use 'from math import *' This loads ALL of the function names into the namespace. You may see this in Zelle or other places so I wanted you to see it here, but I recommend against using it. It loads things we will not need into our code and can lead to confusion as you will have to avoid EVERY function and constant name that the math module uses. It's best to avoid it.

```
# This is a comment
#
# Author: Lars
#
# Play with modules

from math import pi, sqrt, ceil

a = 34
b = -12
c = 34.12

print(sqrt(34))
print(pi)
print(ceil(c))
```

```
5.830951894845301
3.141592653589793
35
```

Python Methodologies for Data Sciences

Remember, you can always look up how functions work by checking out the online documentation. Reading and understanding these documents is crucial to becoming a proficient programmer so start looking at them.

You do not have to understand everything that is there yet (you won't, no worries) but you should start becoming familiar with these online docs.

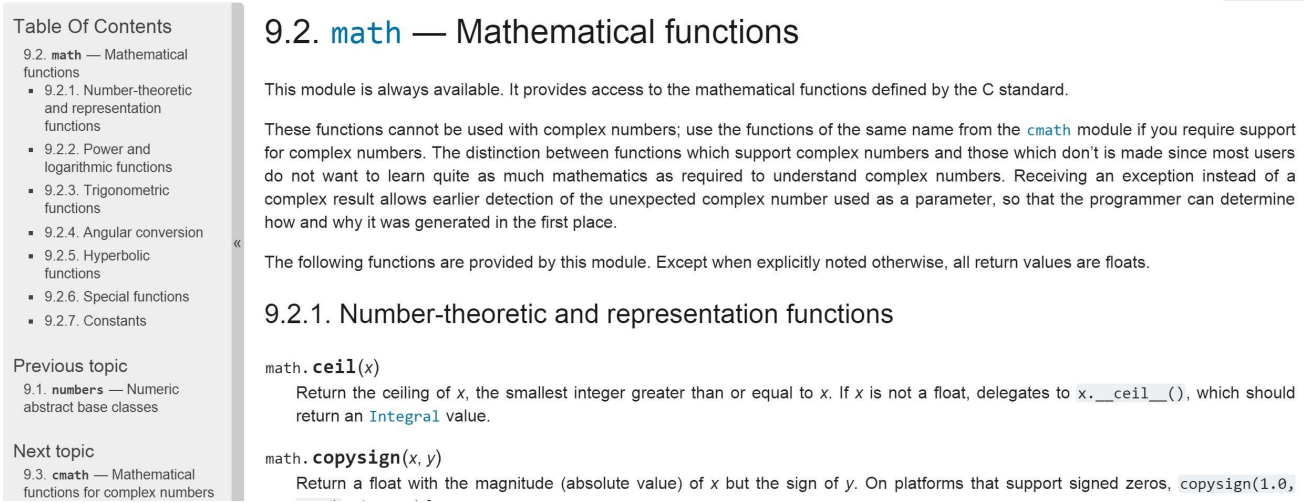


Table Of Contents

9.2. `math` — Mathematical functions

This module is always available. It provides access to the mathematical functions defined by the C standard.

These functions cannot be used with complex numbers; use the functions of the same name from the `cmath` module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don't is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

9.2.1. Number-theoretic and representation functions

`math.`**`ceil`**(*x*)

Return the ceiling of *x*, the smallest integer greater than or equal to *x*. If *x* is not a float, delegates to `x.__ceil__()`, which should return an `Integral` value.

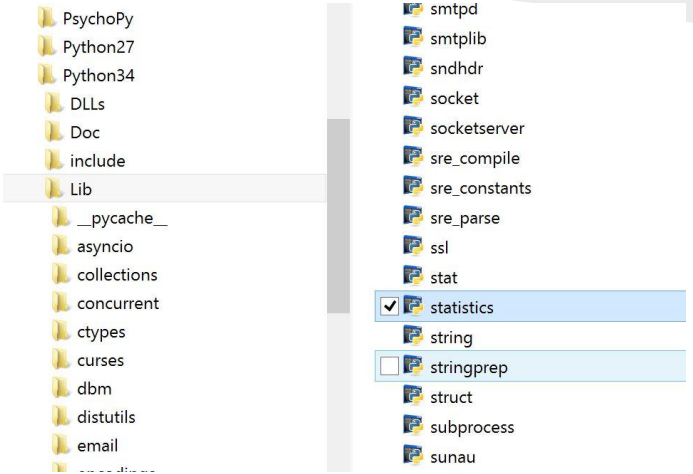
`math.`**`copysign`**(*x*, *y*)

Return a float with the magnitude (absolute value) of *x* but the sign of *y*. On platforms that support signed zeros, `copysign(1.0,`

https://docs.python.org/3/library/math.html

Python Methodologies for Data Sciences

More Behind the Curtain



The code for the standard library modules is actually on your hard drive! When Python sees an import command it looks for the code in three places. First, it will look in the directory where your program is (we'll rely on this later when we write our own modules). Second, it reads a special "environment' variable that your OS (Windows, Linux, MAC) keeps called PYTHONPATH. This variable keeps a list of directories that Python should use for code. Lastly, installations will have a special place for Python libraries. I show a snip of a Windows library above. On LINUX systems the directory is usually usr/local/lib/python.

Python Methodologies for Data Sciences

Because the code is on your hard drive you can look at it anytime you want! This is a great way to learn about programming and see what the module functions you are using are actually doing behind the scenes. Here I look at the 'mean' function in the statistics module. Since we are a programming class leaning towards Data Science, let's go use the statistics package...

```
72 def mean(data):
73     """Return the sample arithmetic mean of data.
74
75     >>> mean([1, 2, 3, 4, 4])
76     2.8
77
78     >>> from fractions import Fraction as F
79     >>> mean([F(3, 7), F(1, 21), F(5, 3), F(1, 3)])
80     Fraction(13, 21)
81
82     >>> from decimal import Decimal as D
83     >>> mean([D("0.5"), D("0.75"), D("0.625"), D("0.375")])
84     Decimal('0.5625')
85
86     If ``data`` is empty, StatisticsError will be raised.
87     """
88     if iter(data) is data:
89         data = list(data)
90     n = len(data)
91     if n < 1:
92         raise StatisticsError('mean requires at least one data point')
93     return _sum(data)/n
```

Python Methodologies for Data Sciences

A quick look at the library documentation for the statistics module shows me some of the functions I can use.

I have mean, mode, median. I even have spread functions so I can get a variance and a standard deviation if I have a population (or a sample). That means I have what I need to do Z scores, yes?

Let's do it...

9.7.1. Averages and measures of central location

These functions calculate an average or typical value from a population or sample.

`mean()`	Arithmetic mean ("average") of data.
`median()`	Median (middle value) of data.
`median_low()`	Low median of data.
`median_high()`	High median of data.
`median_grouped()`	Median, or 50th percentile, of grouped data.
`mode()`	Mode (most common value) of discrete data.

9.7.2. Measures of spread

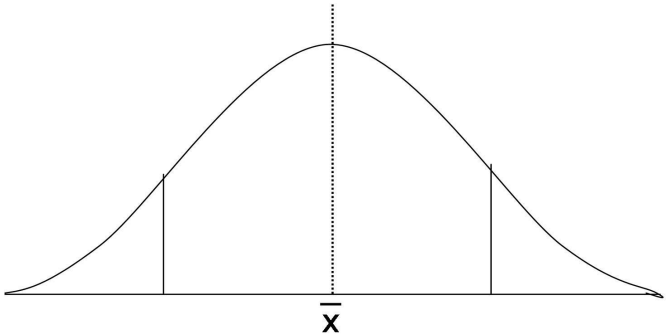
These functions calculate a measure of how much the population or sample tends to deviate from the typical or average values.

`pstdev()`	Population standard deviation of data.
`pvariance()`	Population variance of data.
`stdev()`	Sample standard deviation of data.
`variance()`	Sample variance of data.

https://docs.python.org/3/library/statistics.html

Python Methodologies for Data Sciences

I first import the statistics library. Then I print some values. Then, by using the mean and the standard deviation I can produce Z scores for individual list items to see how far away they are from the mean.




```
# This is a comment
#
# Author: Lars
#
# Use the statistics library


import statistics

myList = [ 12, 23, 34, 2, 45, 65, 7, 34, 12, 34, 5]

print(statistics.mean(myList))
print(statistics.mode(myList))
print(statistics.median(myList))
print(statistics.pvariance(myList))
print(statistics.pstdev(myList))

# Now I have the tools to make a Z score

z = (myList[2] - statistics.mean(myList))/ statistics.pstdev(myList)
print("The Z score for",myList[2],"is",z)

z = (myList[3] - statistics.mean(myList))/ statistics.pstdev(myList)
print("The Z score for",myList[3],"is",z)
```

Python Methodologies for Data Sciences

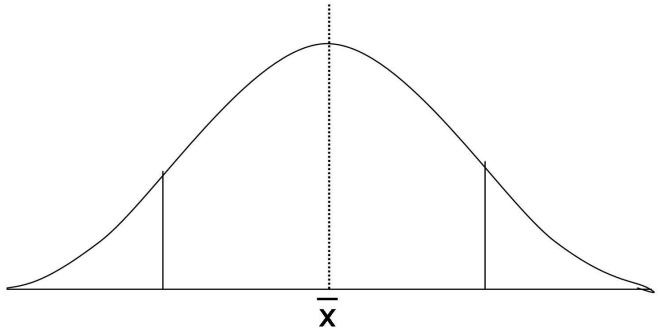
```
# This is a comment
#
# Author: Lars
#
# Use the statistics library


import statistics

myList = [ 12, 23, 34, 2, 45, 65, 7, 34, 12, 34, 5]

print(statistics.mean(myList))
print(statistics.mode(myList))
print(statistics.median(myList))
print(statistics.pvariance(myList))
print(statistics.pstdev(myList))

# Now I have the tools to make a Z score

z = (myList[2] - statistics.mean(myList))/ statistics.pstdev(myList)
print("The Z score for",myList[2],"is",z)

z = (myList[3] - statistics.mean(myList))/ statistics.pstdev(myList)
print("The Z score for",myList[3],"is",z)
```

```
24.818181818181817
34
23
348.87603305785126
18.678223498444687
The Z score for 34 is 0.49157877260558225
The Z score for 2 is -1.2216462566732782
>>>
```

My results (seen below in blue) make sense. Think about it. The mean is around 25, so that would be at the center of the bell curve. The third item is 34, which is 9 away from the mean. Seeing as the standard deviation is 18.6, a Z score of .49 makes sense, I'm about .5 standard deviations from the mean (9 is half of 18).

The same holds true for the second Z score. With the '2' I am 23 to the left of the mean, so I am more than a standard deviation (18.6) away from the mean and I get a Z score of -1.22.

Now we can do simple frequentist stats without SPSS or R. All we need is Python.



Python Methodologies for Data Sciences

Now we look at the time library. I am only going to use one function from it (go check out the others) but it lets me time my operations.

I use the time function to get the current time as the program runs. This is pretty powerful because now I can begin to examine how long it takes my programs to run.

By using this you can begin to see how truly fast computers are.

16.3. `time` — Time access and conversions

This module provides various time-related functions. For related functionality, see also the `datetime` and `calendar` modules.

Although this module is always available, not all functions are available on all platforms. Most of the functions defined in this platform C library functions with the same name. It may sometimes be helpful to consult the platform documentation, b semantics of these functions varies among platforms.

An explanation of some terminology and conventions is in order.

- The *epoch* is the point where the time starts. On January 1st of that year, at 0 hours, the "time since the epoch" is zero. F epoch is 1970. To find out what the epoch is, look at `gmtime(0)`.
- The functions in this module may not handle dates and times before the epoch or far in the future. The cut-off point in t determined by the C library; for 32-bit systems, it is typically in 2038.
- **Year 2000 (Y2K) issues**: Python depends on the platform's C library, which generally doesn't have year 2000 issues, sir and times are represented internally as seconds since the epoch. Function `strptime()` can parse 2-digit years whe format code. When 2-digit years are parsed, they are converted according to the POSIX and ISO C standards: values mapped to 1969–1999, and values 0–68 are mapped to 2000–2068.

https://docs.python.org/3/library/time.html

Python Methodologies for Data Sciences

Here I grab some code that acts as a generator (behind the curtain from Unit 2, yes) for Fibonacci numbers.

If memory serves this code was used for Euler 25. I needed to find the index of the first Fibonacci number with 1000 digits.

I grab the time, perform my work, and then grab the time again, this time minusing the start time to get the elapsed time for my operation.

I created and examined 4782 Fibonacci numbers in about a tenth of a second. Starting to get the idea? Fast.

```
# OK, now we can generate fibs, lets test
start = time.time()
c = 0
for i in fibby(5000):
    c += 1
    #print(len(str(i)))
    if len(str(i)) >= 1000:
        print("The answer is ", c)
        break

elapsed = time.time() - start
print("The answer was found in", elapsed,"seconds")
# I did some testing to ballpark the seed.  4782 is what we get in less than
# a second...
```

```
>>>
The answer is  4782
The answer was found in 0.12013888359069824 seconds
>>>
```

Python Methodologies for Data Sciences

Before we discuss creating our own modules we need to discuss one more thing, third party libraries. Often, especially with Python, the most popular modules are not found in the standard library. They are created by third party programmers and released for use in the Python community. These libraries need to be installed on your systems just like when you installed Python, but once installed you can import their code just like you imported from the standard library.

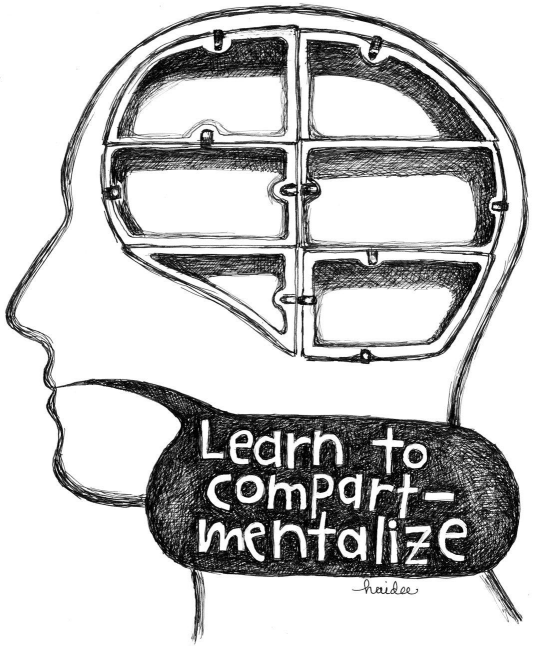
Some of these libraries have become part of every Python programmers tool kit. The four I show to the right are just a sample. Pygame is used to make video games (I will be showing you some of this in the coming weeks), Numpy and Scipy are no-brainers for scientific computing tasks and matplotlib is used for data visualization (we will be using this in unit 4).

Python Methodologies for Data Sciences

Using built in functions, code from the Standard Python library and even code from third party libraries can save us a lot of time and be very convenient, but code reuse is not the only benefit of modularity and compartmentalization.

It can also serve to help us organize our own code and begin to create libraries of our own. It allows us to create files full of code that **we** know we will reuse. Code that is specific to us.

I am going to run through a simple example to show you what I mean.

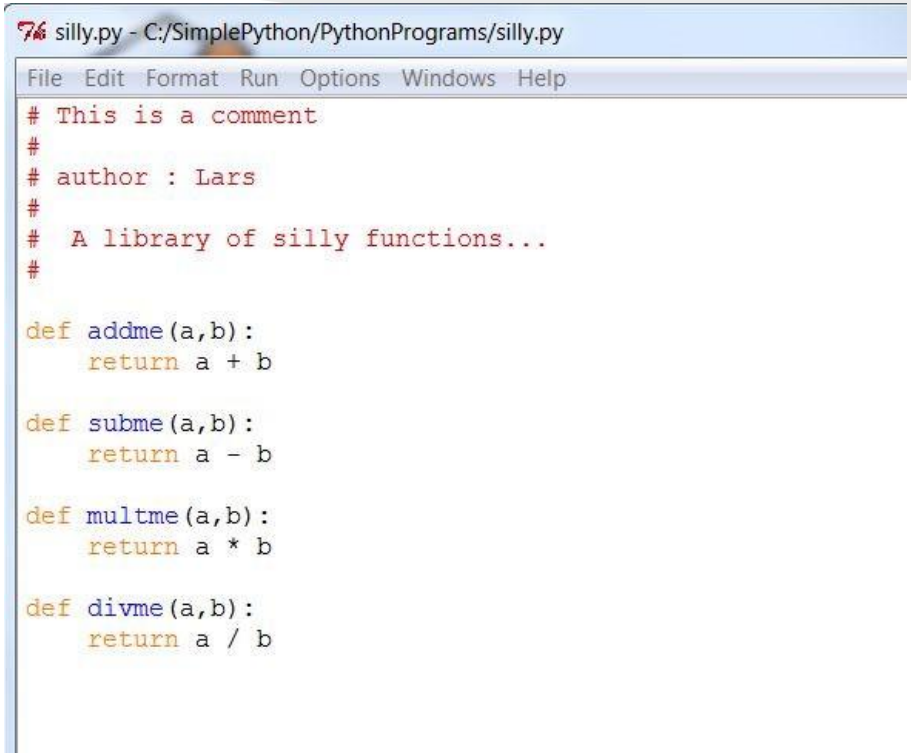


Python Methodologies for Data Sciences

Here I create a simple file (from here forwards known as a module. That's what Python calls it, so when in Rome...) called 'silly.py'

It contains four simple math functions. If you were to run this code nothing would happen (try it). This is just a list of functions waiting to be used.

Code this up for yourself.



Python Methodologies for Data Sciences

Now I create a new module. It contains two void functions that just print a header and a footer to the screen.

I save this (in the same directory as silly.py) as sillyheads.py

Code this up too. Stay with me, it'll make sense in a second...

sillyheads.py - C:/SimplePython/PythonPrograms/sillyheads.py

File Edit Format Run Options Windows Help

```
# This is a comment
#
# author : Lars
#
#  a library of silly header functions...
#

def printheader():
    print("These are some math answers for my report")
    print("****************************************")

def printfooter():
    print("****************************************")
    print(" That's all folks!!")
```

Python Methodologies for Data Sciences

Now it will all come together.

I create a function. It's called doMath(). The first thing I do is import from my two previous modules. I only import what I am going to use (addme and divme, not multme and subme).

I then call printheader(). Then I have two print functions that use my math functions inside of them. I then call for my footers. Now, this is all in a function. **In order to run the program I need to run the function.** I do this at the bottom of the program with the doMath() function call.

Create this program. Call it usesilly.py. Place it in the directory with the two modules you created previously.

usesilly.py - C:/SimplePython/PythonPrograms/usesilly.py

File Edit Format Run Options Windows Help

```
# This is a comment
#
# author : Lars
#
#  Using a library of silly functions...
#

def doMath():
    from silly import addme, divme
    from sillyheads import printheader, printfooter

    printheader()
    print("the answer to 3 + 4 is : ", addme(3,4))
    print("the answer to 12 / 4 is: ", divme(12,4))
    printfooter()


# main program
doMath()
```

Python Methodologies for Data Sciences

usesilly.py - C:/SimplePython/PythonPrograms/usesilly.py

File Edit Format Run Options Windows Help

```
# This is a comment
#
# author : Lars
#
#  Using a library of silly functions...
#

def doMath():
    from silly import addme, divme
    from sillyheads import printheader, printfooter

    printheader()
    print("the answer to 3 + 4 is : ", addme(3,4))
    print("the answer to 12 / 4 is: ", divme(12,4))
    printfooter()


# main program
doMath()
```

This is the result. The code to the left uses the two modules we produced and does not have to type them all over again. We created two of our own modules, imported functions from both (you are not limited to using only one library at a time) and used them in a program.

```
>>> ================================ RESTART ====
>>>
These are some math answers for my report
*****************************************
the answer to 3 + 4 is :  7
the answer to 12 / 4 is:  3.0
*****************************************
 That's all folks!!
>>> ================================ RESTART ====
>>>
```

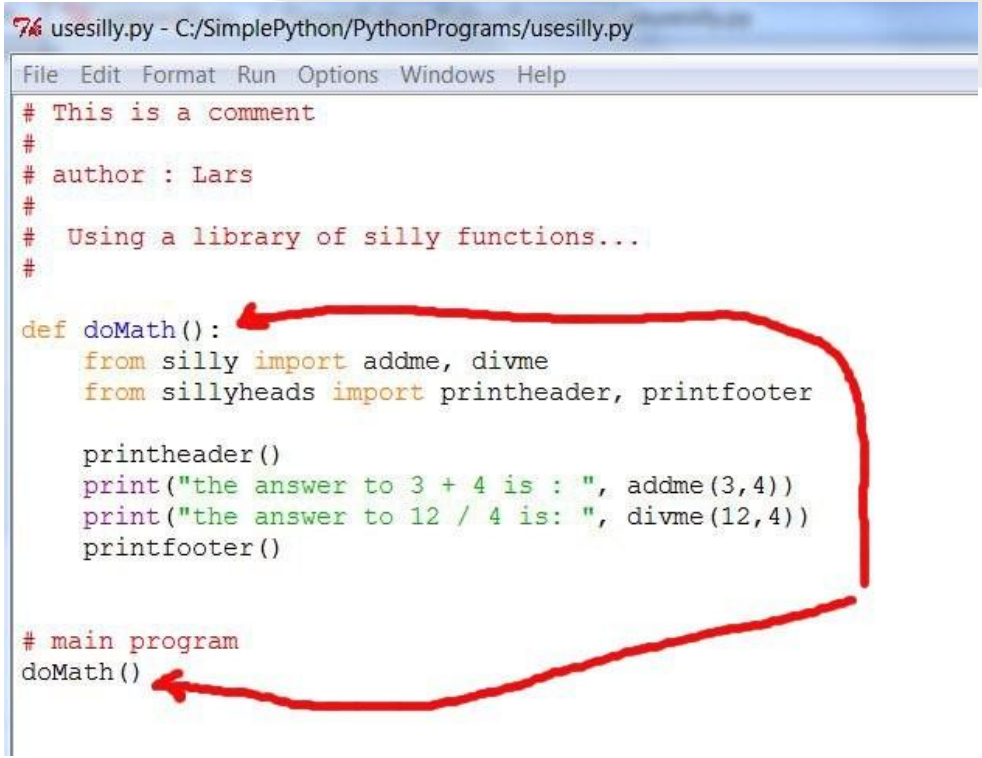
Python Methodologies for Data Sciences

One explanation remains.

Why did we wrap our program code in the form of a function?

I know you have seen this quite a bit in Zelle. He likes to write programs in “main()” functions and then run them with a main() call as the last item of a program.

There’s a very simple reason why we do this...



Python Methodologies for Data Sciences

usesilly.py - C:/SimplePython/PythonPrograms/usesilly.py

File Edit Format Run Options Windows Help

```
# This is a comment
#
# author : Lars
#
#  Using a library of silly functions...
#

def doMath():
    from silly import addme, divme
    from sillyheads import printheader, printfooter

    printheader()
    print("the answer to 3 + 4 is : ", addme(3,4))
    print("the answer to 12 / 4 is: ", divme(12,4))
    printfooter()


# main program
# doMath()
```

useusesilly.py - C:/SimplePython/Python

File Edit Format Run Options Windo

```
# comment

import usesilly

usesilly.doMath()
```

```
These are some math answers for my report
*****************************************
the answer to 3 + 4 is :  7
the answer to 12 / 4 is:  3.0
*****************************************
 That's all folks!!
>>>
```

We do this because if we comment out the call to main() (see the code to the left) we have just taken our ‘program’ and turned it into a ‘module.’ What I mean to say is that now we can actually import ‘usesilly’ into another program and use it. Look above. I create a program called useusesilly.py and just run the doMath() function. The results are the same as when I ran it previously. If this were a report I ran often I just took a lot of work and created a structure that produces my report in one line of code.

Python Methodologies for Data Sciences

silly.py - C:/SimplePython/PythonPrograms/silly.py

File Edit Format Run Options Windows Help

```
# This is a comment
#
# author : Lars
#
#  A library of silly functions...
#

def addme(a,b):
    return a + b

def subme(a,b):
    return a - b

def multme(a,b):
    return a * b

def divme(a,b):
    return a / b
```

sillyheads.py - C:/SimplePython/PythonPrograms/sillyheads.py

File Edit Format Run Options Windows Help

```
# This is a comment
#
# author : Lars
#
#  a library of silly header functions...
#

def printheader():
    print("These are some math answers for my report")
    print("****************************************")

def printfooter():
    print("****************************************")
    print(" That's all folks!!")
```

usesilly.py - C:/SimplePython/PythonPrograms/usesilly.py

File Edit Format Run Options Windows Help

```
# This is a comment
#
# author : Lars
#
#  Using a library of silly functions...
#

def doMath():
    from silly import addme, divme
    from sillyheads import printheader, printfooter

    printheader()
    print("the answer to 3 + 4 is : ", addme(3,4))
    print("the answer to 12 / 4 is: ", divme(12,4))
    printfooter()


# main program
# doMath()
```

useusesilly.py - C:/SimplePython/Python

File Edit Format Run Options Windo

```
# comment

import usesilly

usesilly.doMath()
```

From useusesilly.py I called the code in usesilly.py. From here I called the code in silly.py and sillyheads.py. In this way we have compartmentalized, created headers, footers and math functions that can be reused, and codified an often used process (assuming this was useful and NOT silly) so we can perform it often and with only one command. Powerful stuff.

To sum things up

Modularity or compartmentalization keeps distinct operations separate. This helps us to stay organized and it keeps code from getting too long and complicated.

Remember to use good names for your modules and to keep in mind that you may use the code you are writing in many different programs so keep things as general as you can while still getting the job done.

This is how professional programmers operate, they rarely write complete self-contained programs. They will write modules for themselves and other programmers to use.

Python Methodologies for Data Sciences

To sum things up

We have a world of code at our fingertips when we consider the Python Standard library. We can import these functions and constants and use them in our own code. Make sure you are comfortable with using the Python online documentation:

https://docs.python.org/3/library/

Down the road we will see how we can work with 3rd party libraries when we use matplotlib to create graphs and visualizations with our data.

You can also make your own modules to organize your code and possibly reuse it down the line as we did with all of our ‘silly’ modules and code.

Python Methodologies for Data Sciences

Finally!!!

We're here. Welcome. You are now Python Programmers. We have covered the basics of imperative (like a recipe, remember) programming. You have the rudiments. We're halfway home. Take your time absorbing these slides. Think back to what we have done in Units 1 and 2. You now have the simple components of computer programming under your belt.

Data
Sequences
Iteration
Functions
Modularity

The basics are indeed under the belt, but there is still much to learn. We'll worry about that later though. Right now we celebrate.

Python Methodologies for Data Sciences

Other cool resources

Byte is one of the best intro to Python books on the web. You can use the website or you can look on the SAKAI site for a PDF version that I put there.

You cannot get enough practice. Books for beginners are all different and you will learn something new every time you look at one. I guarantee it. I have been programming for thirty-five years and I learn things when I review these books. They are worth a look.

Practice, practice, practice...

http://www.swaroopch.com/notes/python/

A Byte of Python

Swaroop C H – swaroop@swaroopch.com

Table of Contents

I know, I know...

Again, my eyes were bigger than my stomach (and I have a big stomach ;-)) when I wrote the syllabus.

As I did with dictionaries in the past Unit, I made a promise in the syllabus that I did not deliver on in these slides.

We will cover Exceptions and Exception handling in the next unit. We will likely do dictionaries there as well. It makes sense as these two topics are a bit on the advanced side and will fit nicely in a Unit on Computer Science considerations. That, and if I do not do dictionaries soon Eric will kill me, he really wants you guys to know about them because he know how important they are to software development.

I will likely do a section next unit called ‘orphans’ and we’ll handle the things that were left behind in the first half of the course.

Python Methodologies for Data Sciences

A few simple exercises

1. Write a function called sphereArea(radius) that takes a radius and returns the area of a sphere with that radius.

2. Rewrite Euler 6 using functions; Specifically, create functions for the sum of squares (sos) and the square of the sums.

3. Write a function called sumN(n) that returns the sum of the numbers from 1 to n

4. Write a function called nthFib(n) to compute the nth Fibonacci number. Create a file that stores the first 200.

5. Create a list of the first 1000 multiples of 17. Pickle the list. Then retrieve it from the file you saved it to and print the value to the screen.