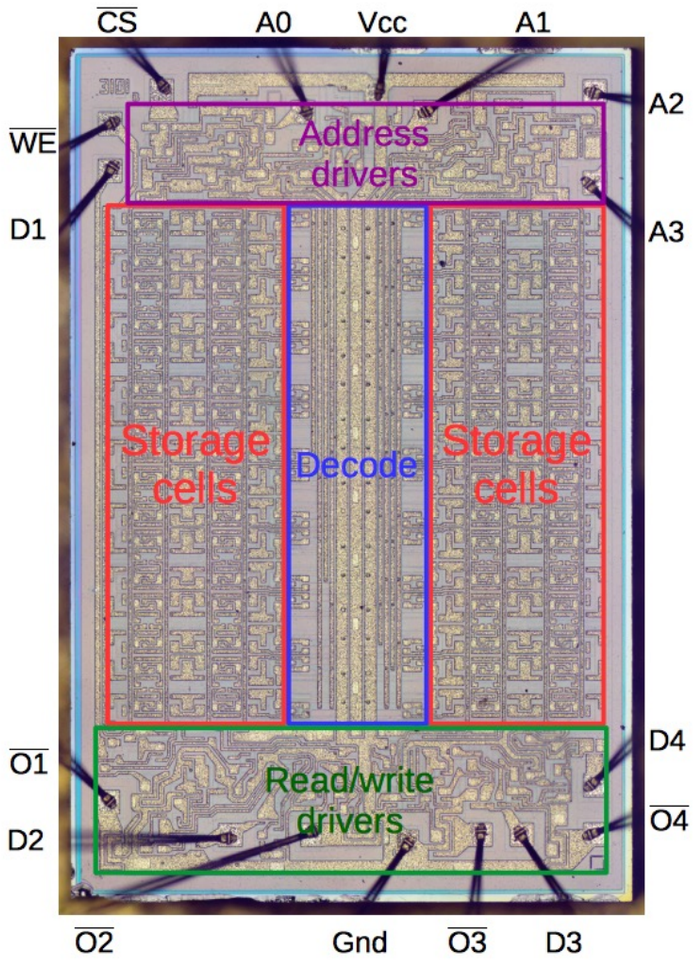


W200 Introduction to Data Science Programming

Starting out with Python

Week 02

Summer 2021

Agenda

UC Berkeley
SCHOOL OF INFORMATION

- 1. Recap of last week and questions
- 2. Running Python
- 3. Expressions
- 4. Objects
- 5. String Objects (& Breakout)
- 6. Control of Flow (& Breakout)

1. Checking in …

UC Berkeley
SCHOOL OF INFORMATION

- Are your environments set up okay?
  - Homework assignments are posted on GitHub "upstream" on Monday evenings
  - TA grade and provide feedback on homework; Instructors may review; Instructors grade & provide feedback on projects
- Make sure you're using Python 3.x (3.6 higher).
- GitHub ready to go?
- Bash info: [http://hyperpolyglot.org/unix-shells]
- There are optional intros to Unix commands in the Optional Resources folder.

1. Ask your questions

UC Berkeley
SCHOOL OF INFORMATION

Reminder of communications channels

Check out the TAs and OH.

Syllabus Link

Live Calendar

Homework Assignments

Email instructors & students: w200-python-2021-summer@googlegroups.com

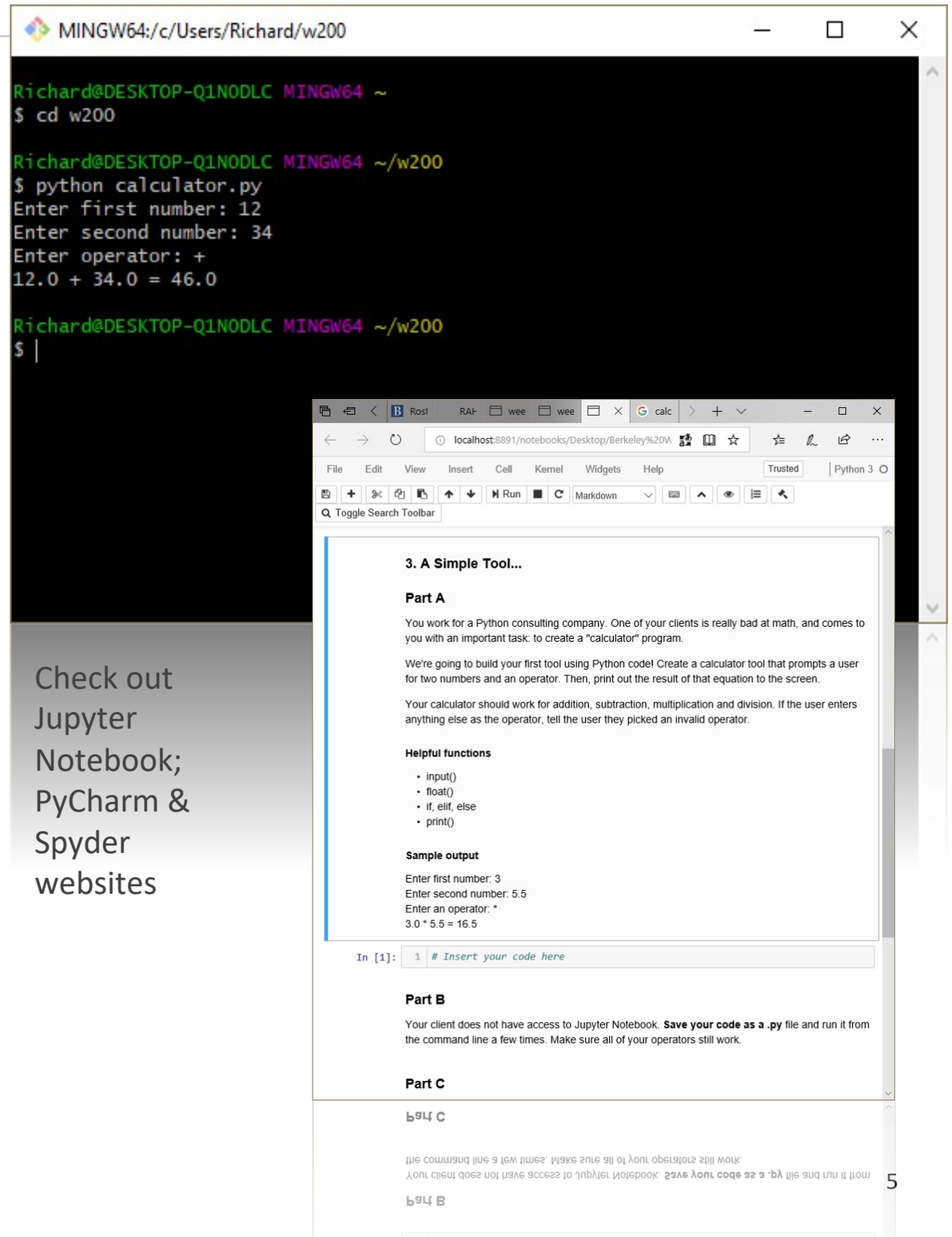
Email instructors only: mids-python-instructors@googlegroups.com

Slack Channels: ucbischool.slack.com channel #w200-python

2. Running Python (reminder)

UC Berkeley
SCHOOL OF INFORMATION

- Command line
  - Use a text editor [Atom, BBEdit, NotePad++; and save as a .py file. You *may* need to specify >python3 myfile.py if you have multiple versions installed.
- Jupyter Notebook
  - App that interprets and provides error msgs. Useful for testing - be sure to reload variables and remember that instructors don't have access to your hard drive.
- Be sure to restart the kernel and re-run your homework before you submit it.



Check out Jupyter Notebook; PyCharm & Spyder websites

3. Expressions

UC Berkeley
SCHOOL OF INFORMATION

Building blocks reminder.

Assignment		`x = 5`
Addition	`+`	`x + 5`
Subtraction	`-`	`x - 5`
Division	`/`	`x / 5`
Multiply	`*`	`x * 5`
Exponent	`**`	`x ** 5`
Cumulative	`+=`	`x += 5` `x = x + 5`

Equivalency		`x == 5`
Cumulative	`-=`	`x -= 2`
Integer division	`//`	`x // 5`
Modulus	`%`	`x % 2`

4. Concepts about Objects

UC Berkeley
SCHOOL OF INFORMATION

More about objects later but let's start with some principles.

- **What is an "object"?**
  - a programmatic way of expressing things in the physical world* of the time you'll have to discover answers on your own (that's how it is in computing).
- **For OOP languages …**
  - the "object" is the parent, from which child objects are copied and used; these child copies are the instantiation of the Object. In coding the "Object" becomes a "Class."
- Everything in Python is an object. **All objects** …
  - (a) contain all their variables and behaviors [methods] [encapsulation],
  - (b) can accept multiple parameters [polymorphism], and
  - (c) can be copied and new properties added [inheritance].
- Communication between **Objects isn't necessarily linear**: they can send messages to/from each other, given appropriate triggers.
- In OOP there's a hierarchy of libraries; going up the chain of a command leads ultimately to the base Object; from there we can traverse a chain back down to another type of object (hence "type-casting").

* Rather like Plato's Theory of forms, *De re publica,* Book 10.

```
# a coin-toss program
import random  # an external library

class Coin:
     # defining our own classes
     # a magic method python has for constructing an object in memory
     def __init__(self):
          self.sideup = "Heads"

     # toss method generates a random number in the range of 0-1.  if 0,
     # then sideup is heads; else it's tails.

     def toss(self):
          if (random.randint(0, 1) == 0):
               self.sideup = “Heads"
          else:
               self.sideup = "Tails"

     # create a function to get the value
     def get_sideup(self):
          return self.sideup


# define the MAIN function
def main():
     # create a variable (the object “Coin” from the Coin class).
     my_coin = Coin()

     # display the side of the coin that’s facing up
     print("This side is up:", my_coin.get_sideup())

     # toss that coin!
     print("I'm tossing the coin … ")
     my_coin.toss()

     # show the results …
     print("This side is up: ", my_coin.get_sideup())

# all done creating the 'legos' of our code, now let's start the show…
main()
# end of the file.
```

Class

variables

methods()

1) “construct” the Coin; set to “heads”

2) define actions (“methods”) that do something with the Coin (here, just “toss”)

3) more methods: find out what side is up get_sideup()

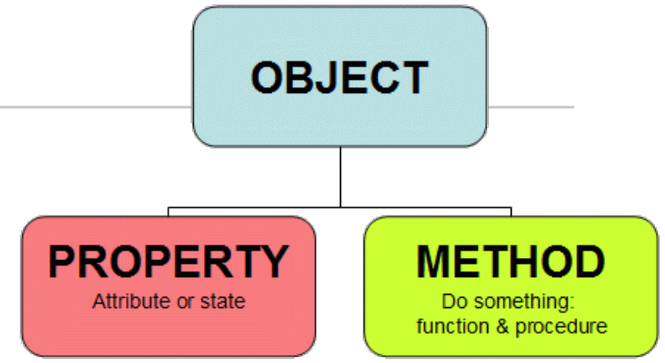
Use “main” to organize events.
Define a door into the program (main).
Create the “Coin” (so we have one to play with).
Use the pre-defined actions **methods** on the coin instance.

OPTIONAL ENRICHMENT

4. Objects

conception from ideas into executable code

UC Berkeley
SCHOOL OF INFORMATION



Objects reflect "real-world" phenomena. Here is my cat … expressed in computer code as a Cat object.

4. Objects have types

UC Berkeley
SCHOOL OF INFORMATION

Everything in python is an object; we can get info about the object as well as its values.

- Python lets us assign values and we can test what data type we're using,
  e.g., `x = 5` → `type(x)` → returns `int`

```
>>> x = 5
>>> type(x)
<class 'int'>
>>>
```

- Boolean [bool, true/false; 0/1]
- Integers (`int`) counting numbers
- Floating point (`float`), e.g., 3.14159
- String (`str`), a sequence of contiguous characters, e.g., [c][a][t]

Type cast: convert data from one type to another, e.g., float(x)

```
st = input("Enter some text: ") #Default is string
fl = float(input("Enter data: ")) #Convert to float
```

```
# -*- coding: utf-8 -*-

""" a little nested if script """
""" since python defaults to ASCII unless otherwise informed ... we like UTF-8 """

""" can you get a loan to work remotely in Tahiti?  I sure would like to! """

min_salary = 50000.0          # min salary to get loan
min_years = 2                 # min years on the job

# get the annual salary
salary = float(input("Enter the annual salary: $ "))

# years on the job
years_on_job = int(input("and the number of years on the job? "))

# check conditions
if salary >= min_salary:
    if years_on_job >= min_years:
        print("\n\tJoy!  Tahiti here we come!\n")
    else:
        print('Sorry! The minimum is ', min_years,' to qualify.')
        print("Treat yourself to a consolation peña colada on us.")

else:
    print("Annual income must be at least $", format(min_salary, ',.2f')," to qualify.")
```

Notes:

- Encoding directive utf-8. Often automatically added by some IDEs.
- Python “”” style comments.
- Defining some global variables (`min_salary` and `min_years`)
- Get input from the stdin (keyboard) and converting the input into a `float` or `int` type.
- Nested-if statements. Notice the range checking, use of escape-sequences `\n\t` and format commands.
- `print()` sends data to the currently selected stdout.

OPTIONAL ENRICHMENT

5. Namespaces | Object space

UC Berkeley SCHOOL OF INFORMATION

Think of the variable name as what we call it; but that var can point to different values in RAM the object space

The concept of a "namespace" refers to the active set of variables, as we name and use them in our code. Using a namespace usually keeps variable names unambiguous. Common in SQL, C++, etc., referring to the ***scope & visibility*** of a variable.

In python, we can have several variables of different names share the namespace … but they can all point to the *same value* in *object space*.

For students with experience in other languages, see the parallels:

SQL Example

SQL (Entity SQL) uses "namespaces" to avoid conflicts for global identifiers, such as type names, entity sets, functions.

E.g.: a database has 3 tables, A, B, and C. Each table has a "record_number" field. To distinguish them we use the table . field:

```
A.record_number = 50
B.record_number = 233
```

C++ example

```
// Here we can see that more than one variables
// are being used without reporting any error.
// That is because they are declared in the
// different namespaces and scopes.
#include <iostream>
using namespace std;

// Variable created inside namespace
namespace first
{
    int val = 500;
}

// Global variable
int val = 100;

int main()
{
    // Local variable
    int val = 200;

    // These variables can be accessed from
    // outside the namespace using the scope
    // operator ::
    cout << first::val << '\n';

    return 0;
}
```

OPTIONAL ENRICHMENT

5. Namespaces | Object space

vari-able

name space

object space

Variables go into a special object called a “name space.” This is related to ”variable scope".

var | **name space** | **object space**

`x` — x ⋯⋯⋯⋯→ 5 → *int*

`b_int` — b_int → 36 → *int*

`c_int` — c_int → 36 → *int*

`greet` — greet → print(“Bonjour”) → *function*

file

But without a namespace we can’t reach elements in the object space

6. Objects: Strings

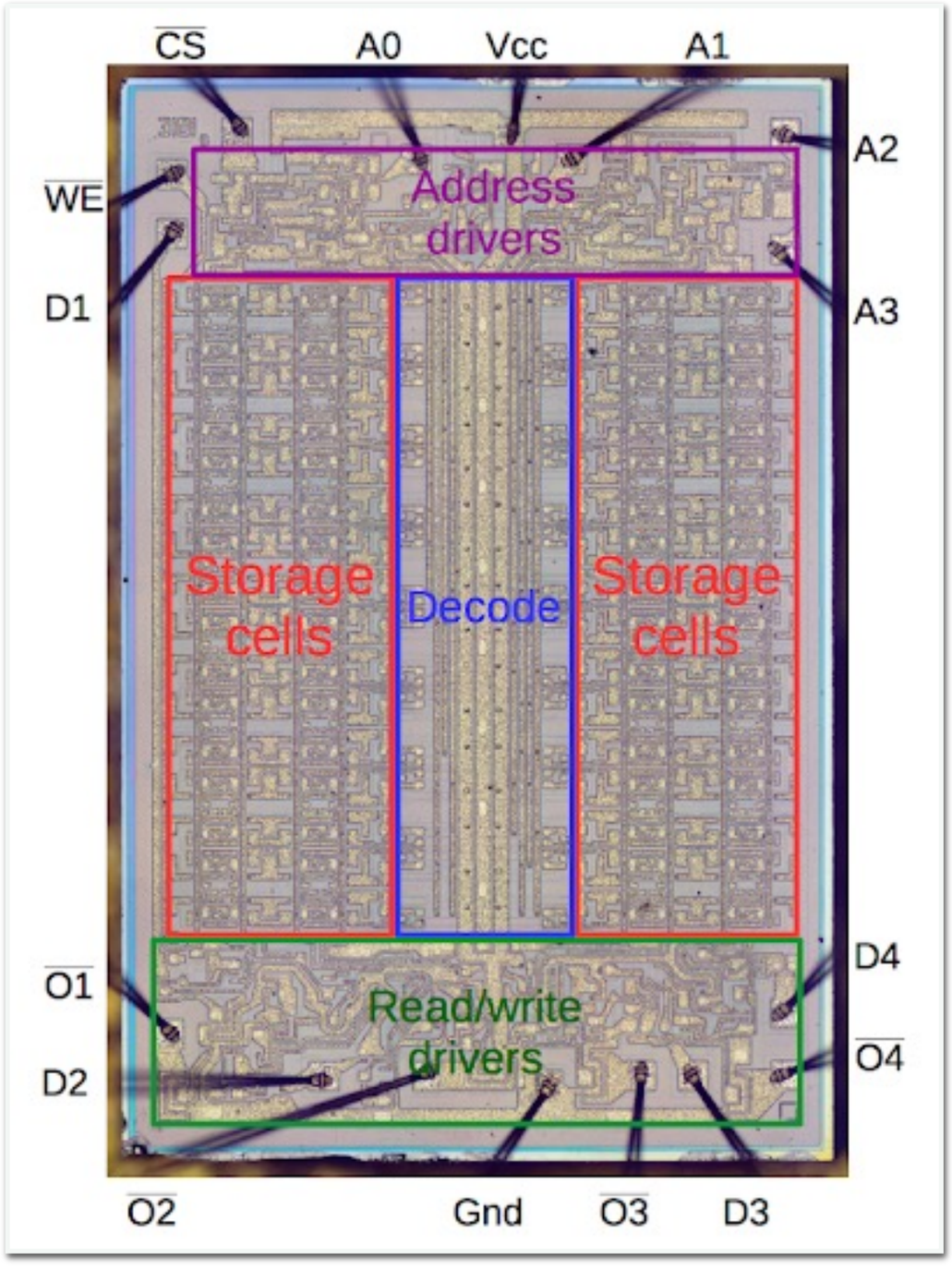
UC Berkeley
SCHOOL OF INFORMATION

Once we learn the syntax of an object, it's easy to anticipate its methods.

- A String is a "sequence object" [imagine the memory address of this string is 0 and is n bytes long. - Locate entire strings, individual letters, range of letters]

- Strings can be joined (or "concatenated") and manipulated:
  1. concatenation: `"cat" + "dog"` outputs "catdog"
  2. multiplication of chars: `"-"*5` outputs `-----`
  3. *<string>*`.upper()`, e.g., `x = "cat"  print(x.upper())`
  4. *<string>*`.lower()`
     1. s = input("Type here:")
     2. int_cast = int(input("write a number: "))

Python gains a *lot* of speed by using "pointers" and like many languages, the pointers are hidden from the programmer. [No issues of malloc and referencing points!] Notice that as an OOP language, Python often uses String methods (`.` and `()`) to copy, adjust, and then replace the object in RAM, using the same variable name.



Optional: A side note.

RAM (random access memory) looks like a grid. Every wire on the x-axis that crosses the wire on the y-axis creates a memory address. There are two registers: address and data.

The data are stored by being *encoded* and *decoded (*pointers and dereferencing pointers; may include casting object into other data type, often as String).

So … when a Python variable (object) is stored in RAM, our variable "point" to the object's memory address in RAM. A *copy* of the variable has a different name but *still points to the same memory address*. This is important.

If we have `a = 5` and then `b = a`, both a and b refer (point) to the same memory address. Consequently, changing the value of "b" changes the value of "a".

This might cause a problem in your coding. We'll see this addressed in the idea of "scope and visibility" of our variables and in certain Python commands, such as `copy` versus `deepcopy`.

(But it is cool how our code is so intimate with the chip!)

Strongly recommended: read
Sipser, M., *Introduction to the theory of computation.*
(3rd ed). A pdf earlier version is here.

http://www.righto.com/2017/07/inside-intels-first-product-3101-ram.html

OPTIONAL ENRICHMENT

6. String Objects

UC Berkeley SCHOOL OF INFORMATION

Python is fairly loosely typed; note "straight" quotes versus "typographers' quotes".

- Single- and double quotes:
  - Python, like other loosely-typed languages, allows using single- and double-quotes around Strings. Keep in mind that often when concatenating data we use a mix of " and ' … the use of these *must* be balanced, else a hard to find error.
- Comments in python code use *three* " , e.g., " " "
- \ means an "escape sequence", note: `\n, \r\n, \t`

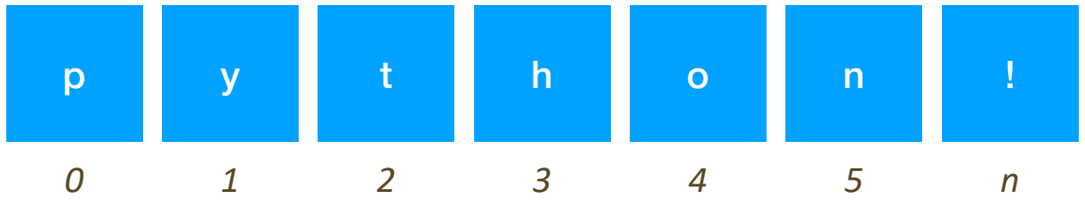
```
' '       # single quotes (the dagger kind)
"  "      # double quotes (notice the pairing)

t = input("Welcome, you're here! ")
          # double quotes surrounding string
          # so the single quote is safe
s = input('Welcome, you\'re here!')
          # single quotes, so middle one needs
          # to be "escaped"
```

5. String Objects: Slicing

UC Berkeley
SCHOOL OF INFORMATION

Slicing, like many python commands, has lots of shortcuts.



`[ 0 ]`	from the start of the string
`[ -1 ]`	one letter from the end of the string
`[ 0:3 ]`	from the start of the string to the 4th letter
`[ 1:-1 ]`	from 1 to the next to last letter
`[ 1:5:2 ]`	from 1 to 5, by 2 *at a time*
`[ :-1 ]`	beginning (:) to second to last letter (*n* - 1)
`[ : ]`	entire string
`[ ::-1 ]`	entire string, reversed

"Slicing" is surprisingly important in data sci and processing of data, during "data cleansing."

We rely in slicing to test data before ingesting, to test data for certain conditions …

Breakout Activities:

UC Berkeley
SCHOOL OF INFORMATION

Tribbles, Spiders, and Calculator

1. Use what you downloaded for week 2 in Jupyter to create a string variable that prints exactly

```
The “trouble with
Tribbles” is that they
\\\EAT/// too many MREs.
```

2. Using one line of Python code, make your variable from part 1 print Tribbles backwards “`selbbirT`” 300 times.

```
selbbirT selbbirT selbbirT …
```

6. Flow Control: if, while

UC Berkeley
SCHOOL OF INFORMATION

Review of the basics

if statements &
nested if statements

```
if x > 2:
    print(“x is greater than 2”)
elif x < 0:
    print(“x is negative”)
else:
    print(“x is less than 2 but still positive”)
```

while loops

```
countdown = 5
while countdown > 0:
    print(countdown)
    countdown -= 1
print(“Blast off!”)
```

We’ll see several other techniques next week, such as `for x in range()`*, when we work with lists and dictionaries.*

An example of using for loops in various functions … basic Python [1 of 2]

```
""" a for-loop demo """

""" for-loops are a count-controlled lop, iterating sequentially over a
      set of data, e.g., an array or other set of data """

def aboutThisScript():
      print("-"*60)
      print("This script will iterate over a set of integers, Strings, \nand then "\
            "use a function in the iteration for new values ")

def allDone():
      print("\nCheck out the script for little demos.  That's it.  Cheers.\n")

def iterateInts():
      print("-"*60)
      for i in [1, 3, 5, 7, 9, 13, 17]:
            print("Prime number: ", i)

def iterateStrings():
      print("-"*60)
      for name in ['Tom', 'Ming Wa', '明哇', 'विक्रम', 'Rex', 'Fatima', 'Kelly' ]:
            print(name)


def squareValues(i):
      return i^2

def iterateFunc():
      print("-"*60)
      print("Squaring values from a list.\n")

      list_1 = [6, 3, 1, 4, 5, 1]

      for i in list_1:
            print("\t", squareValues(i) )
```

OPTIONAL ENRICHMENT

Ultimately we'll move to anonymous, lambda functions … basic Python [2 of 2]

```
def lambdaMapDemo():
      """ notice we're walking from scalars (for loops) to
            vectorization -that is being able to make looping more
            efficient by coding to let python use pointers (variables
            that point to memory addresses holding the actual values);
            we don't use actual pointers as in C++ but we write code
            that let's python do so for efficiency. """
      list_2 = [1, 2, 3, 5, 8, 13, 21 ]

      """ notice the reading from left to right... the source of the data is list_2;
            Then each value is squared.  The lambda "maps" each value sequentially as "x"
            to the calculation; then stores all the results in a NEW list, squaredFib.
      """
      squaredFib = list(map(lambda x: x*2, list_2))

      print("-"*60)
      print("Here is a list of values squared of ", len(list_2), " values.", sep="")
      print(squaredFib)

# ************************************************************
#
#     Above each of the actions are defined in functions; we want, tho, to
#   demonstrate iterating using for loops and various data.  Now that the
#   code pieces "legos" I call 'em are all set, now let's call them in order
#   in our main function.  Finally we invoke main to start the process.
#
# ************************************************************

def main():
      aboutThisScript()

      iterateInts()
      iterateStrings()
      iterateFunc()
      lambdaMapDemo()

      allDone()

# invoke the script starting here.
main()
```

OPTIONAL ENRICHMENT

6. Nested while loops

UC Berkeley
SCHOOL OF INFORMATION

don't worry about perfect spacing

```
row = int(input("Enter an integer: "))

# while row >= 0:

j = 0
while j <= row:
    print(j, end=" ")
    j += 1
```

```
Enter an integer: 5
0 1 2 3 4 5
```

```
Enter an integer: 5
0 1 2 3 4 5
0 1 2 3 4
0 1 2 3
0 1 2
0 1
0
```

```
row = int(input("Enter an integer: "))
# outer loop
while row >= 0:
    j = 0
    #inner loop
    while j <= row:
        print(j, end=" ")
        j += 1
    print(" ")
    row -= 1
```

```
""" a little script showing boolean flag and a while loop """

keep_going = "y"  # a boolean flag - keep going 'til quit. common technique.

# calculate a series of commissions
while keep_going == “y":

    # get sales and commission rate
    sales = float(input("\nWelcome.\n\tPlease enter the sales amount $: "))
    comm_rate = float(input("\tand commission rate: "))

    # calculate
    commission = sales * comm_rate

    # display the commission
    print("Commission is $", format(commission, ',.2f'), sep='')

    """ keep going? as long as the flag is true, keep going.
        note, btw, that by forcing string data to lower()
          we don't have to check
        for Y and y.  Try to validate-on-input to reduce the
          amount of code you need.
    """
    keep_going = input("\nCalculate another? [y|Y any other key to quit.] ").lower()
```

OPTIONAL ENRICHMENT

Breakout Activities: Spiders

UC Berkeley
SCHOOL OF INFORMATION

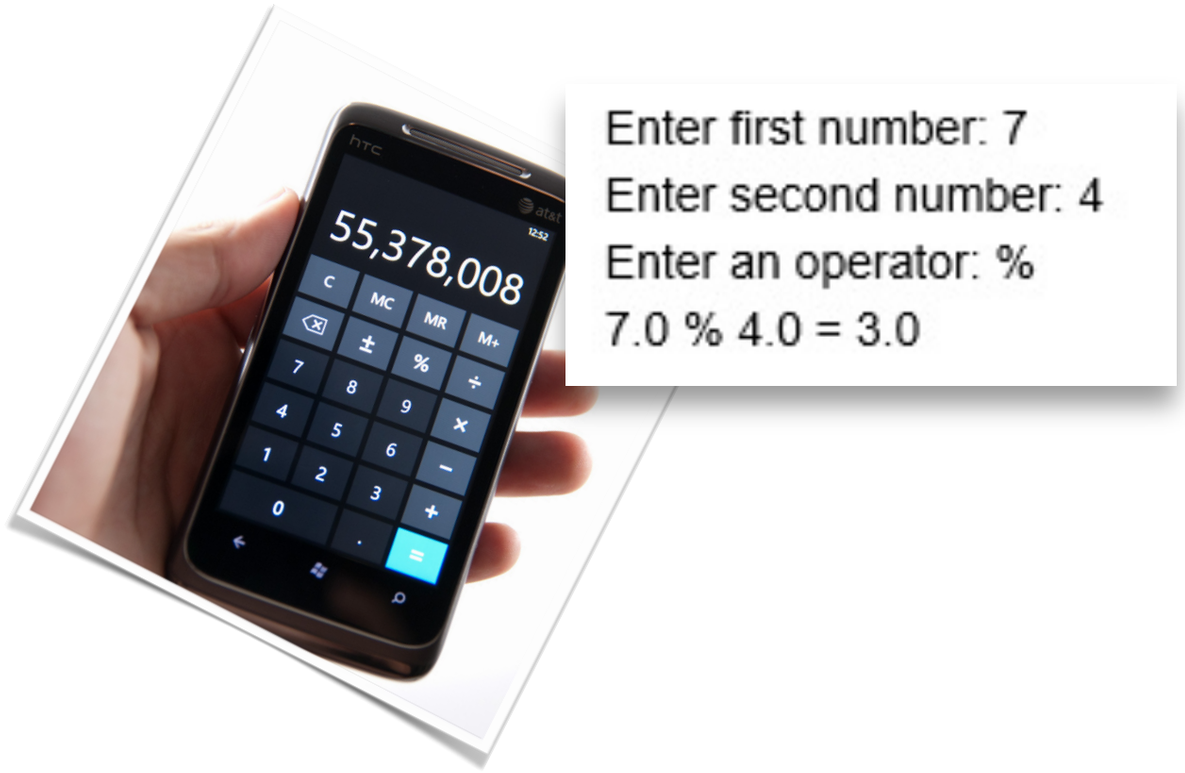
```
Do you have 8 legs?no
Do you have 4 legs?no
You are a bicycle
```

https://www.youtube.com/watch?v=DVAkVwy4TD4

Optional Breakout: Calculator

UC Berkeley
SCHOOL OF INFORMATION

Make a calculator - the purpose is to practice conditions (“if” statements) and saving your work as a .py and executing it.




```
Enter first number: 7
Enter second number: 4
Enter an operator: %
7.0 % 4.0 = 3.0
```

https://www.clayallsopp.com/posts/definition-of-design-terms/

That’s it!

UC Berkeley
SCHOOL OF INFORMATION

- That’s it! Enjoy a great week and keep up with your drills and homework assignments.