SMU | Lyle School of Engineering

CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding

Jan 21 2020, Lecture 1

About Me

- About:
  - Me : Xinyi Ding, PhD student, Computer Science Department
  - Email : xding@smu.edu
  - Home page : xding.me
  - Research Interests: Ubiquitous Computing, Machine Learning, Educational Data Mining

Agenda

- Agenda:
  - Logistics and syllabus
  - About this course
  - Install python
  - First python program
  - Variables and Data types

Syllabus

- Syllabus details: http://xding.me/cs1340/
- Githhub page: https://github.com/dxywill/cs1340
- Assessment:
  - Home Assignments: weekly based (20%)
  - Quizzes: one week notification (10%)
  - Lab projects: bi-weekly (30%)
  - Mid exam: (20%)
  - Final exam: (20%)
- Use canvas for course management

Syllabus

- Lab Sessions:
  - M 10:00AM - 11:50 AM or Tu 1:00PM - 2:50 PM or

Thursday 11:00-12:50PM

  - Start from the 3rd week (Feb 3), Caruth 485

- Office hours: TBD, check canvas for updates
- Help Desk caruth 484.
- Altshuler Learning Enhancement Center (A-LEC)

Syllabus

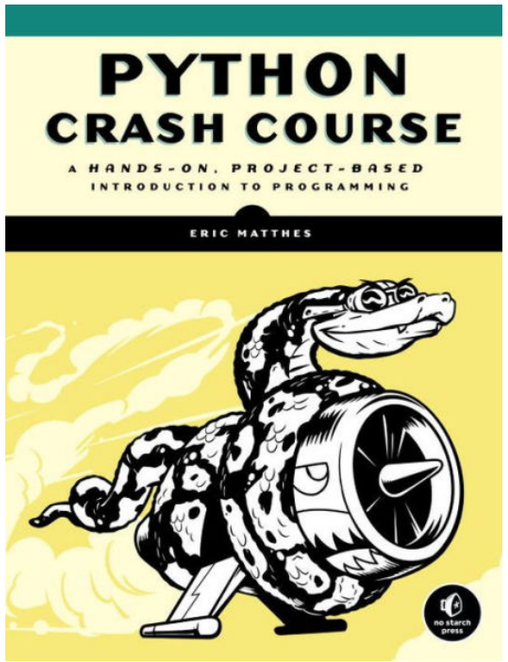
- Participation: required, missing three or more will lead to points loss (Rarely are these measures needed!)
- Late Submission:
  - Home assignments
    - Up to total 10 days without credit reduction (use wisely)
    - Must be submitted within one week after deadline
  - Lab Projects:
    - No late submission will be accepted

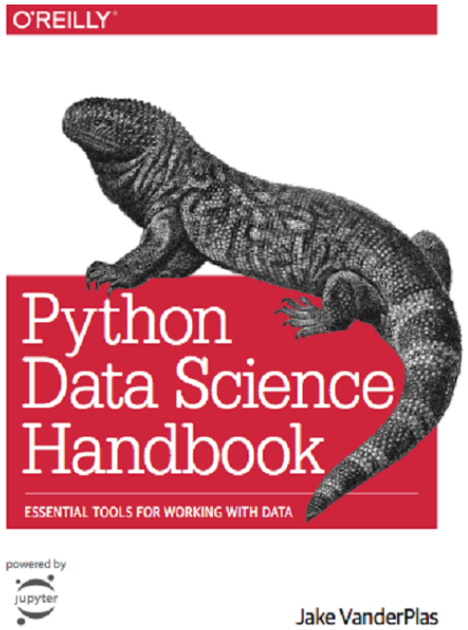
Syllabus

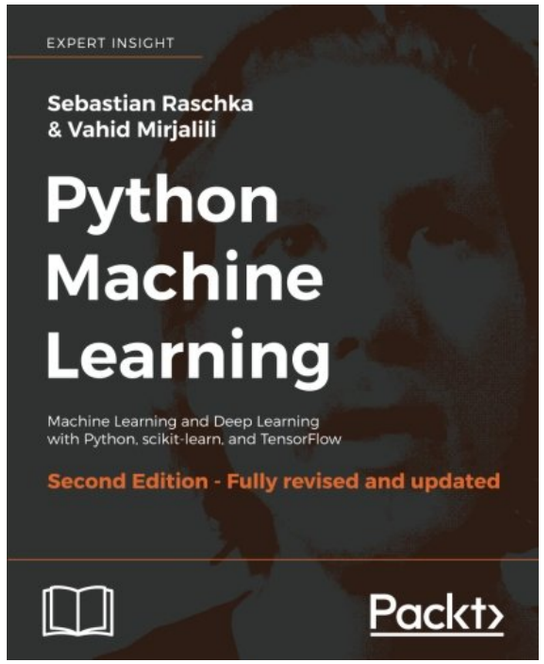
- Disability Accommodations
  - Students needing academic accommodations for a disability must first register with Disability Accommodations & Success Strategies (DASS).
- Religious Observance
  - Religiously observant students wishing to be absent on holidays that require missing class should notify their professors in writing at the beginning of the semester.
- Find more: http://xding.me/cs1340/

Introduction

- Textbooks:
  - **Not required**, but some recommended Books…





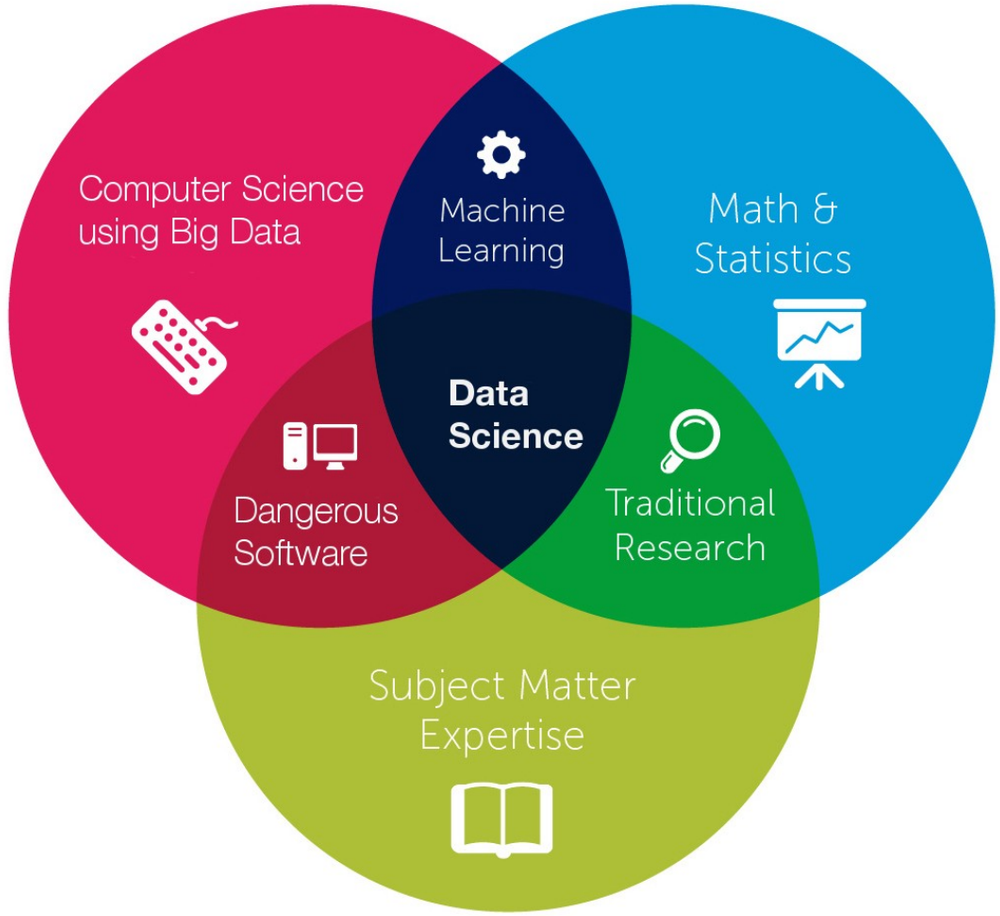


- If you have a specific question, try Google first
- stackoverflow.com
- PEP-8 style guide for python

  https://www.python.org/dev/peps/pep-0008/

Introduction

- Why Python?
  - Easy to learn
  - Efficient and Powerful
  - Big Community support
  - …
- Why Data Science?
  - Data fuels the future
  - Many career options
  - Make the world a better place
  - …



Introduction

- Prerequisite: ***None***
- Course Overview:
  - Part 1:
    - Variables, Data types, Common data structures (List, dict..)
    - Control statement, if/else, loops
    - Functions
    - Object Oriented Programming (OOP)
  - Part 2:
    - IPython, Jupyter Notebook
    - Numpy, Pandas, Matplotlib
    - Scikit-Learn

How to do well in this course?

- Do not fall asleep
- Practice, Practice and Practice
- Have Fun!

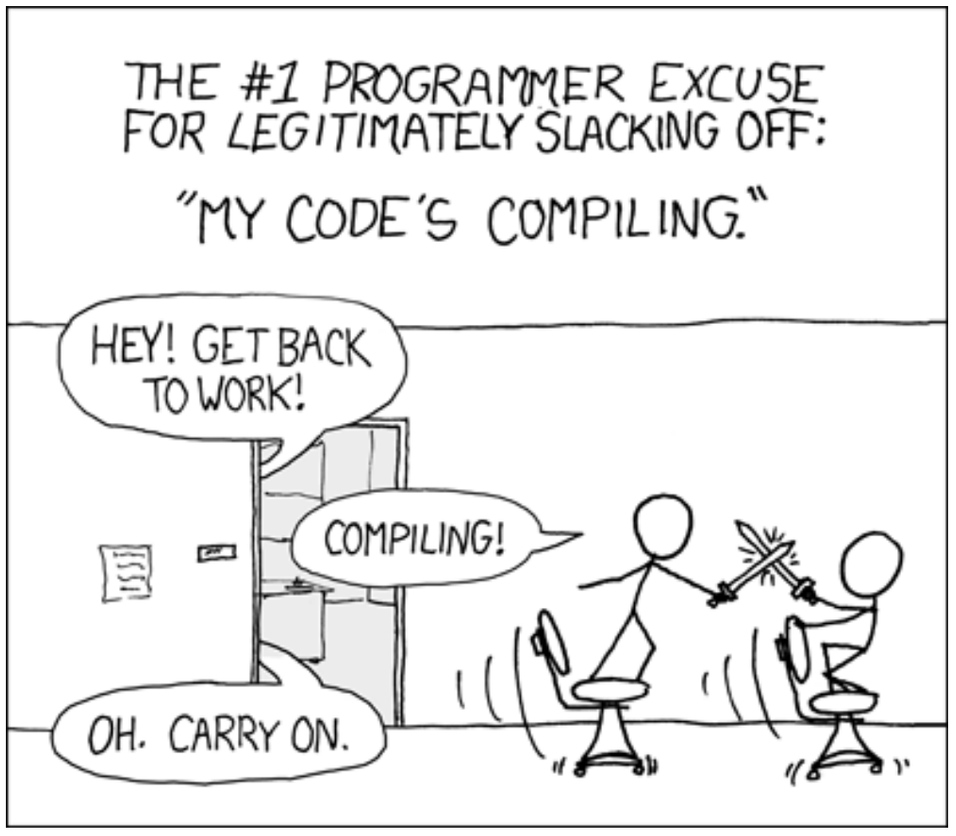
Four stages of morning class

Python programming language

"Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, *Python's design philosophy emphasizes code readability with its notable use of significant whitespace."*

—Wikipedia

- Compiled language:
  - C/C++
  - Delphi
- Interpreted language:
  - Python
  - PHP
  - Ruby



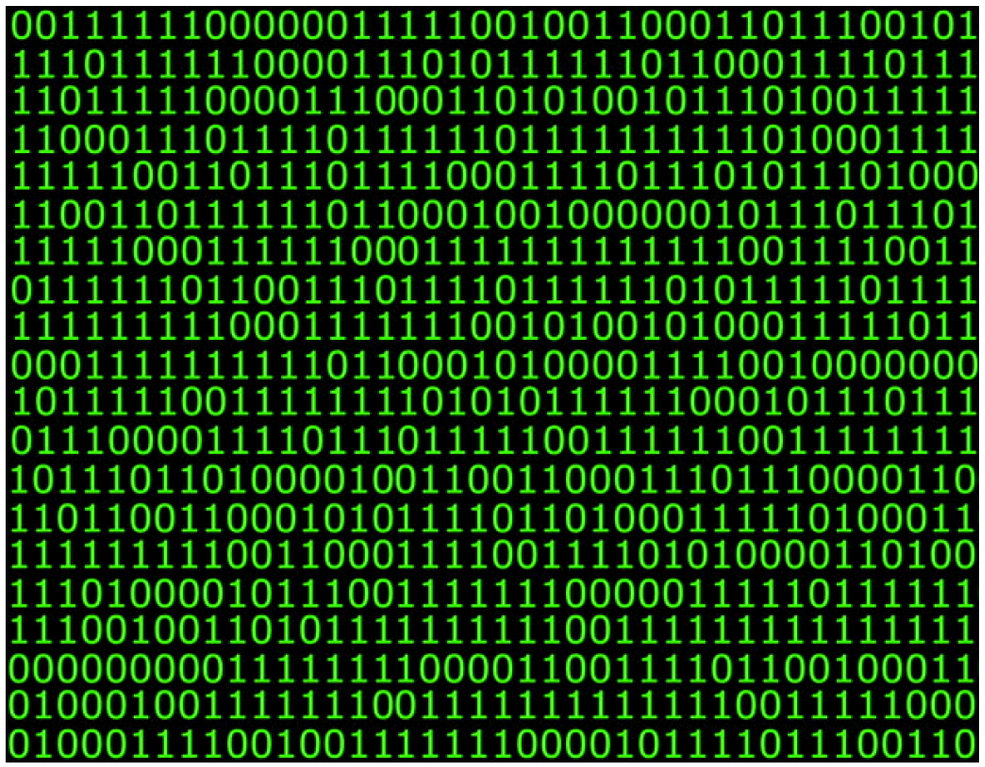
The fact is that the interpreted/compiled distinction is actually an arbitrary one

Python programming language

- Language features
  - Interpreted language
  - Indentation instead of braces
  - Dynamically typed: variables do not have a predefined type
  - Simple Object system
  - Rich, built in collection types:
    - Lists
    - Tuples
    - Dictionaries (maps)
    - Sets

Python programming language

```python
def load_data(fileName):
    rows = []
    max_skill_num = 0
    max_num_problems = 0
    with open(fileName, "r") as csvfile:
        reader = csv.reader(csvfile, delimiter=',')
        for row in reader:
            rows.append(row)
    index = 0
    print("the number of rows is " + str(len(rows)))
    tuple_rows = []
    #turn list to tuple
    while(index < len(rows)-1):
        problems_num = int(rows[index][0])
        tmp_max_skill = max(map(int, rows[index+1]))
        if(tmp_max_skill > max_skill_num):
            max_skill_num = tmp_max_skill
        if(problems_num <= 2):
            index += 3
```



Python Code

Interpreter

Binary Code

Python programming language

- Versions of Python
  - Python 2.X is dying…
  - Python 3.X
    - install through anaconda
    - use conda environment

- Check python version: *python - - version*

```
XinyideMacBook-Pro-4:~ xinyi$ python --version
Python 3.5.2 :: Anaconda custom (x86_64)
XinyideMacBook-Pro-4:~ xinyi$
```

- Check which python in use: *which python*

```
XinyideMacBook-Pro-4:~ xinyi$ which python
/Users/xinyi/anaconda/bin/python
XinyideMacBook-Pro-4:~ xinyi$
```

Anaconda

- Anaconda Python distribution include:
  - Python
  - A lot other useful tools
- Anaconda can help:
  - Installing Python on multiple platforms
  - Separating out different environments
  - Dealing with not having correct privileges and
  - Getting up and running with specific packages and libraries

Anaconda Environment

- conda environment
  - A conda environment is a directory that contains a specific collection of conda packages that you have installed.
    - *For example, you may have one environment with NumPy 1.7 and its dependencies, and another environment with NumPy 1.6 for legacy testing*
- Create conda environment: *conda create --name cs1340*
- Activate conda environment: *conda activate cs1340*
- Other useful commands*:* https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html#managing-environments

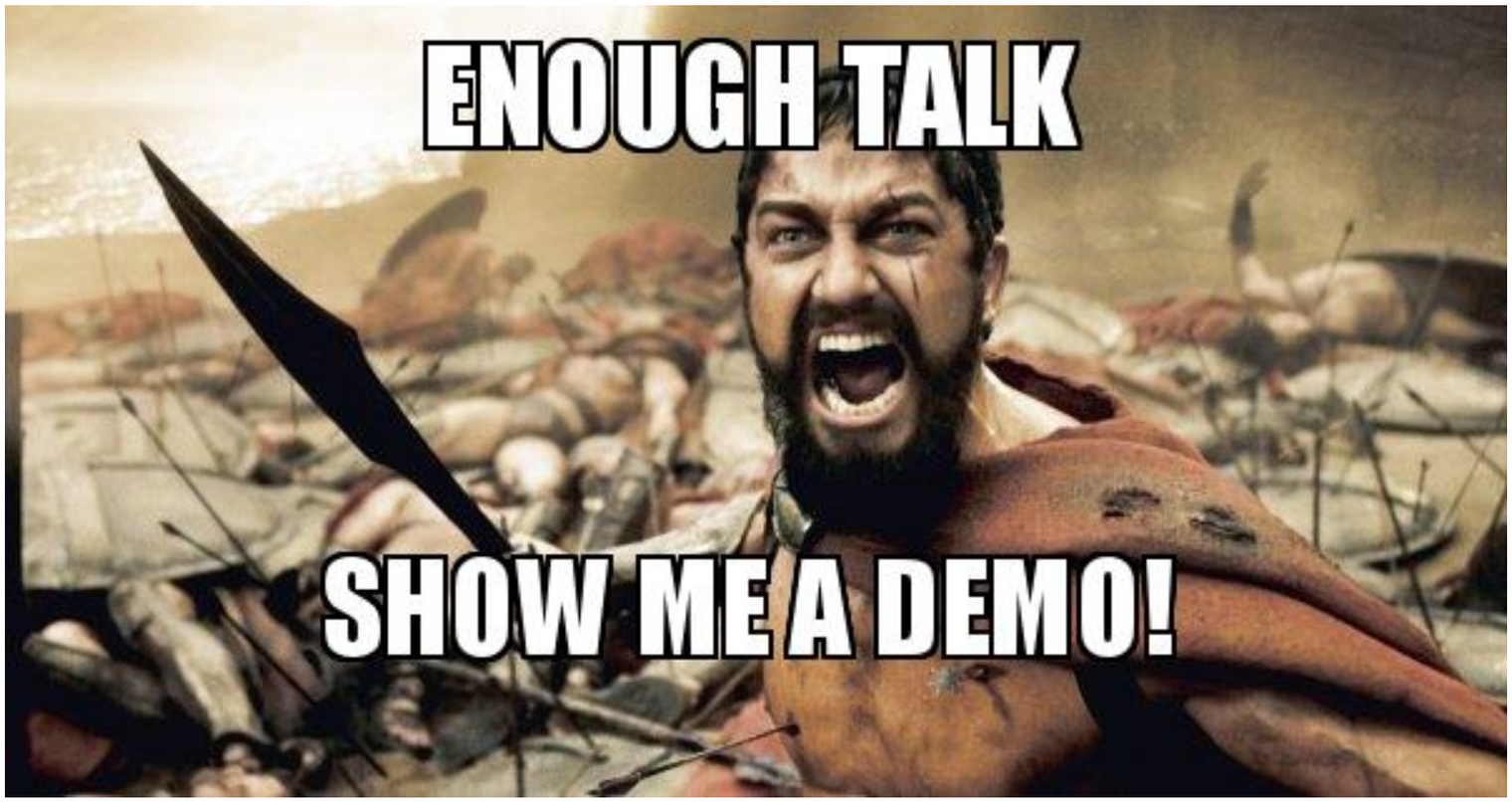
Anaconda Environment

```
(cs1340) XinyideMacBook-Pro-4:~ xinyi$ which python
/Users/xinyi/anaconda/bin/python
(cs1340) XinyideMacBook-Pro-4:~ xinyi$
```

```
(torch) XinyideMacBook-Pro-4:~ xinyi$ which python
/Users/xinyi/anaconda/envs/torch/bin/python
(torch) XinyideMacBook-Pro-4:~ xinyi$
```

```
(mlearn) XinyideMacBook-Pro-4:~ xinyi$ which python
/Users/xinyi/anaconda/envs/mlearn/bin/python
(mlearn) XinyideMacBook-Pro-4:~ xinyi$
```

Demo



Python programming language

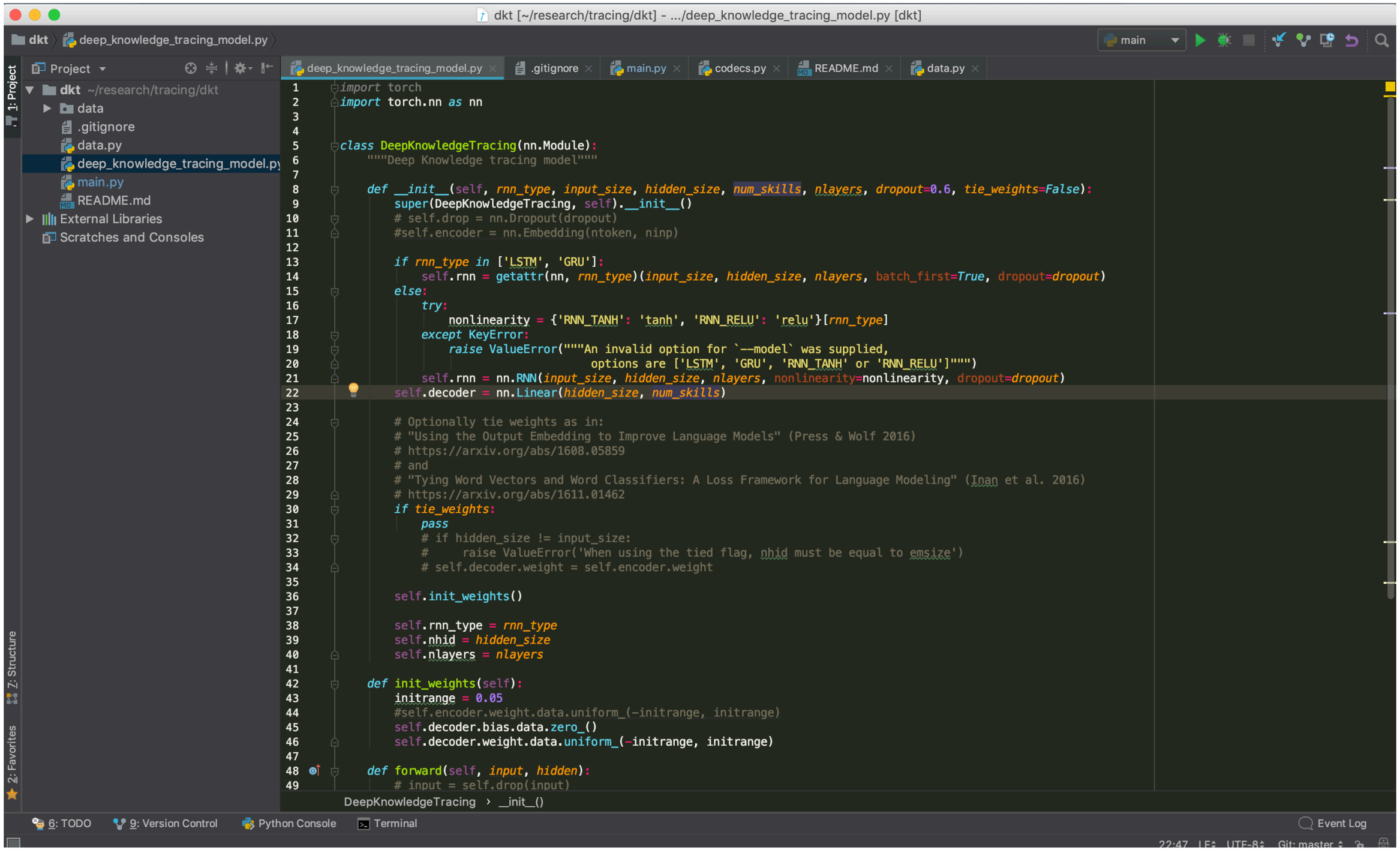
- Ways to use python
  - Interactive shell

```
xinyi — python — 130×39
~ — python
(cs1340) XinyideMacBook-Pro-4:~ xinyi$ which python
/Users/xinyi/anaconda/bin/python
(cs1340) XinyideMacBook-Pro-4:~ xinyi$ python
Python 3.5.2 |Anaconda custom (x86_64)| (default, Jul  2 2016, 17:52:12)
[GCC 4.2.1 Compatible Apple LLVM 4.2 (clang-425.0.28)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello world")
Hello world
>>>
```

  - Python script: *python hello_world.py*

```
week1 — -bash — 130×39
~/Courses/cs1340/week1 — -bash
(cs1340) XinyideMacBook-Pro-4:week1 xinyi$ ls
hello_wolrd.py
(cs1340) XinyideMacBook-Pro-4:week1 xinyi$ python hello_wolrd.py
Hello wolrd
(cs1340) XinyideMacBook-Pro-4:week1 xinyi$
```

Pycharm: Integrated development environment (IDE)



More resources: https://www.jetbrains.com/pycharm/

Python programming language

- A program consists of:
  - Values and types: *4, 'Hello world'*
  - Variables, a name refers to a value
    - an assignment statement creates new variables and give them values
  - Statements, a unit of code that the Python interpreter can execute. We have seen *print and assignment*
  - Expressions, a combination of values, variables and operators, *1 + 1*
  - ...

Variables and Data Types

- Variables
  - containers for storing data values.
  - no command for declaring a variable.
  - created the moment you first assign a value to it.

For example, Create two variables to store age and name

```
age = 5
name = "xinyi"

print(age)
print(name)
```

Variables and Data Types

- Variable Names
  - Variable names can contain only letters, numbers, and underscores.

    They can start with a letter or an underscore, but not with a number.
  - Spaces are not allowed in variable names, but underscores can be used
  - Avoid using Python keywords and function names as variable names
  - Variable names should be short but descriptive.
  - Case Sensitive

For example, first_name, message_1, age are valid names,

but not 1_message, print

Variables and Data Types

- Common Data types
  - Strings
  - Integer
  - Float
  - Boolean

Strings

- Strings
  - A series of characters
  - Can use "" or '' to specify
  - Unmatched can occur within the string
  - Use triple double-quotes for multi-line string

```
'This is a string.'
"This is also a string."

"The language 'Python' is named after a TV show Monty Python, not the snake"

'The language "Python" is named after a TV show Monty Python, not the snake'


""" You can define strings that across multiple lines
like this.

The above line is left empty on purpose
"""
```

Strings Manipulation

- String concatenation

```
first_name = "Xinyi"
last_name =  "Ding"
greetings = "Hello"

message = greetings + " " + first_name + " " + last_name

a_long_string = "duplicate this three times" * 3
```

Numbers

- Integers
  - x = 3
  - y = 12345
- Floats
  - pi = 3.1415926
  - e = 2.71828

Operator	meaning	Examples
+	plus - Add two operands	x+y
-	Minus - subtract right operand from the left	x-y
*	Multiplication- multiply two operands	x*y
/	Division - devide left operand by the right one	x/y
%	Modulus - remainder of the division of left operand by the right	x%y
//	Floor division - division that results into whole number adjusted to the left in the number line	x//y
**	Exponent - left operand raised to the power of right	x**y

Arithmetic operators

Boolean

- The Boolean Type
  - either True or False

```
>>> 3 > 5
False
>>> 1 < 3
True
>>>
```

Comments

- Comments
  - Start comments with # -the rest of line is ignored by the Python interpreter.
  - Python does not support multiline comments like C/C++ or Java.

```
# Say hello to everyone
print('Hello Python people!')
```

However, there is nothing to stop you to use multi-line docstrings as multiline comments.

Comments

- Good comments
  - be short, straight to the point, and add informative value.

- Example of bad comment

```
b = 59                    # assign the value of 59 to b
```

- Example of good comment

```
salestax10 = 1.10         # defining a sales tax 10%
salestax20 = 1.20         # defining a sales tax 20%
```

Whitespace

- Whitespace is meaningful in Python: especially indentation and placement of newlines.
  - Use a newline to end a line of code.
    - Use \ when must go to the next prematurely
  - No braces {} to mark blocks of code in Python...

    Use consistent indentation instead
  - Often a colon appears at the start of a new block
  - (E.g. for functions and class definitions)

Assignment

- Binding a variable in Python means setting a name to hold a reference to some object
  - Assignment creates references, not copies(like java)
- A variable is created the first time it appears on the left side of an assignment expression:
  - x = 3
- Multiple assignment
  - x, y = 3, 5

Agenda

- Agenda:
  - Quick review concepts from previous lecture
  - Sequence data types
  - Lists
  - Common operations on lists

Variables and Data Types

- Variables
  - containers for storing data values.
  - created the moment you first assign a value to it.

- Common data types
  - Strings: *'this is a string'*
  - Integer: *39*
  - Float: *3.1415926*
  - Boolean: *Ture, False*

Sequence types

- When you have millions of items to store

```
employee_1_name = "Alice"
employee_2_name = "Bob"
employee_3_name = "Charlie"
employee_4_name = "David"
employee_5_name = "Eric"

...
```

- Use Sequence types
  - Tuples

```
employee_names = ("Alice", "Bob", "Charlie", "David", "Eric")
```

  - Lists (most often used)

```
employee_names = ["Alice", "Bob", "Charlie", "David", "Eric"]
```

Sequence types

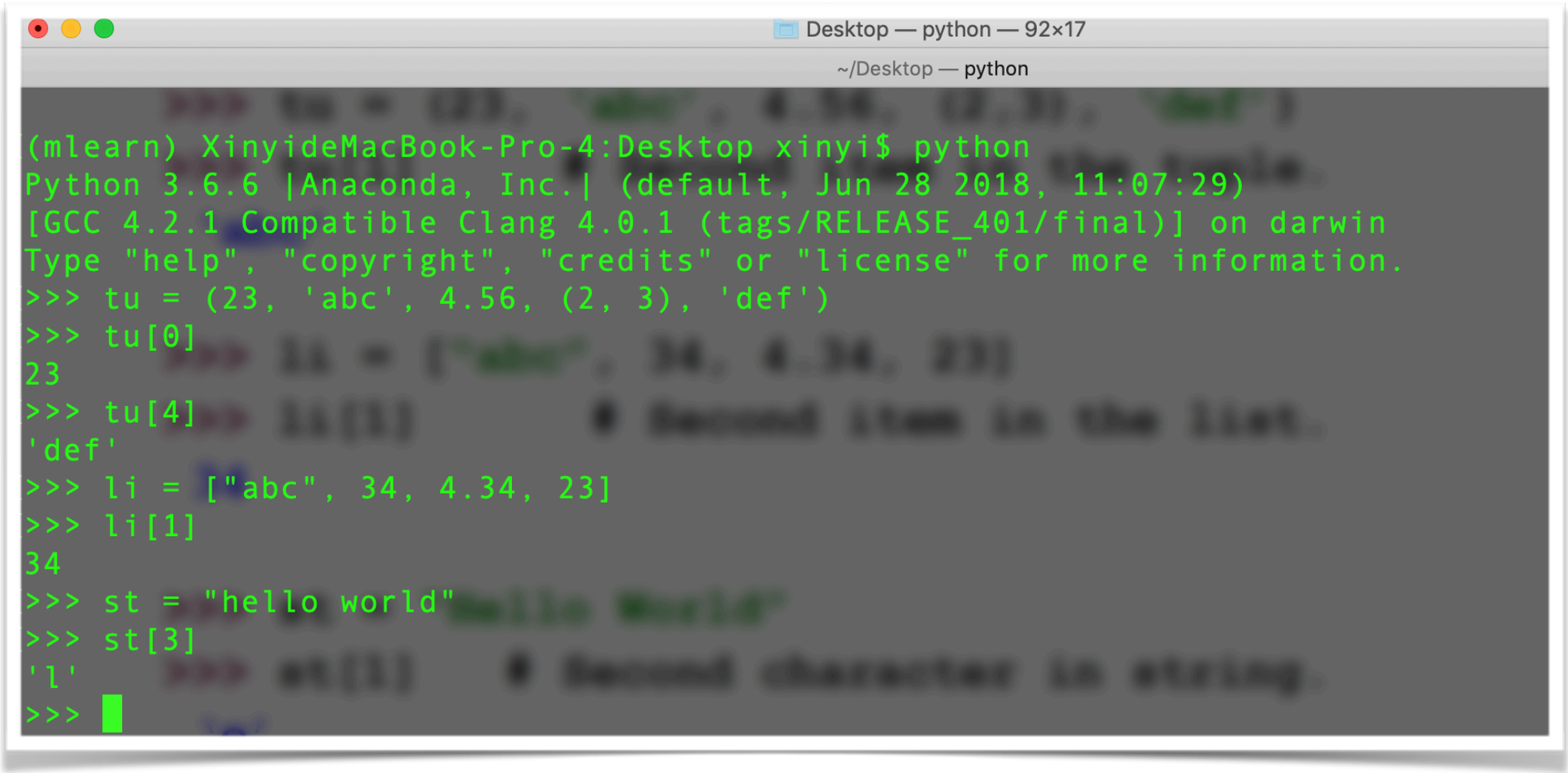
- Tuple
  - A simple **immutable** ordered sequence of items
    - Immutable: a tuple cannot be modified once created....
    - Items can be of **mixed types**, including collection types
- List
  - **Mutable** ordered sequence of items of **mixed types**
- String
  - **Immutable**
  - Conceptually very much like a tuple

Sequence types

- The three sequence types (tuples, strings, and lists) share much of the same syntax and functionality.
  - Tuples are defined using parentheses(and commas).
    - *tu = (23, 'abc', 4.56, (2, 3), 'def')*
  - Lists are defined using square brackets(and commas).
    - *li = ["abc", 34, 4.34, 39]*
- Strings are defined using quotes (", ' , or """).
  - *st = "Hello world"*
  - *st = 'Hello world'*
  - *st = """ This is a multi-line string that uses triple quotes """*

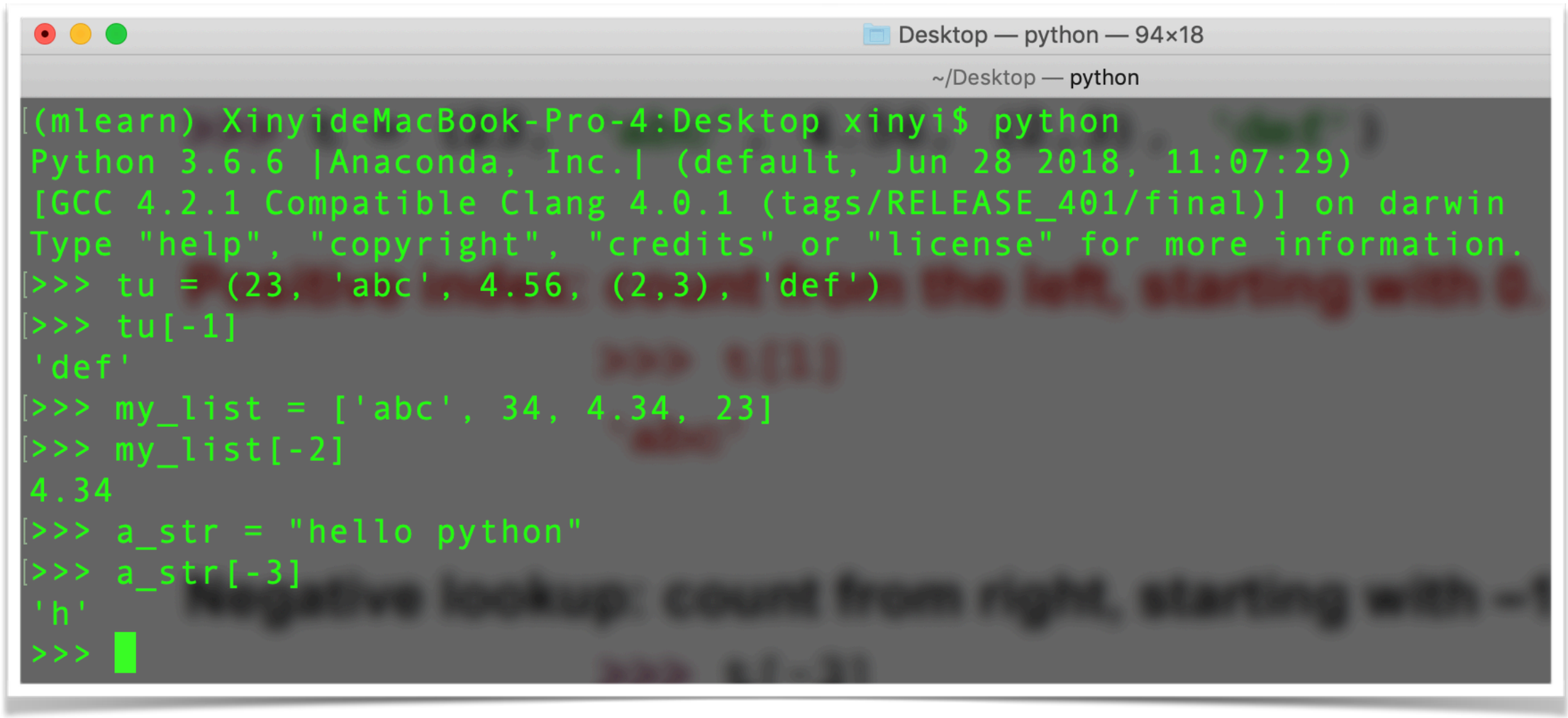
Sequence types

- We can access individual members of a tuple, list, or string using square bracket notation.
- **Note that all are 0 based...**



Negative indices

- Positive index: count from the left, **starting with 0**
- Negative lookup: count from the right, **starting with -1**



Index Errors

- Index Errors
  - when you have three items in your list, but you ask for the fourth element (Python starts indexing at 0, not 1)

```
>>> employee_names = ["Alice", "Bob", "Charlie", "David", "Eric"]
>>> employee_names[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>
```

- A little bit summary, we have seen:
  - *syntax errors, variable undefined errors, index errors.*

Slicing: Return a copy of a subset

- Return a copy of the container with a subset of the original members. Start copying at the first index, and stop copying before the second index.

```
>>> employee_names = ["Alice", "Bob", "Charlie", "David", "Eric"]
>>> employee_names[1:4]
['Bob', 'Charlie', 'David']
>>> employee_names[0:4]
['Alice', 'Bob', 'Charlie', 'David']
>>> employee_names[:4]
['Alice', 'Bob', 'Charlie', 'David']
>>> employee_names[4:]
['Eric']
```

Lists

- **Mutable** ordered sequence of items of **mixed types**
- Use square brackets []
- Individual elements in the list are separated by commas
- Indexing starts from 0, not 1

```
>>> bicycles = ['trek', 'cannondale', 'redline', 'specialized']
>>> print(bicycles)
['trek', 'cannondale', 'redline', 'specialized']
>>>
```

Accessing Elements in a List

```
>>> bicycles = ['trek', 'cannondale', 'redline', 'specialized']
>>> print(bicycles)
['trek', 'cannondale', 'redline', 'specialized']
>>> print(bicycles[0])
trek
>>> print(bicycles[1])
cannondale
>>> print(bicycles[-1])
specialized
>>>
```

Using individual element in a List

```
>>> my_first_biclye = bicycles[0]
>>> message = "My first bicyle was a " + my_first_biclye
>>> print(message)
My first bicyle was a trek
>>> message = "My first bicyle was a " + bicycles[0]
>>> print(message)
My first bicyle was a trek
>>> print("My first bicyle was a " + bicycles[0])
My first bicyle was a trek
>>>
```

Demo

Changing, Adding, and Removing Elements

- Modifying elements in a List
  - To change an element, use the name of the list followed by the index of the element you want to change, and then provide the new value you want that item to have

```
>>> motorcycles = ['honda', 'yamaha', 'suzuki']
>>> print(motorcycles)
['honda', 'yamaha', 'suzuki']
>>> motorcycles[0] = 'mazda'
>>> print(motorcycles)
['mazda', 'yamaha', 'suzuki']
>>>
```

Changing, Adding, and Removing Elements

- Adding elements to a List
  - Appending elements to the end of a list
  - Inserting elements into a list at any position
  - The extend method

```
>>> motorcycles = ['honda', 'yamaha', 'suzuki']
>>> motorcycles.append('bmw')
>>> print(motorcycles)
['honda', 'yamaha', 'suzuki', 'bmw']
>>> motorcycles.insert(0, 'tesla')
>>> print(motorcycles)
['tesla', 'honda', 'yamaha', 'suzuki', 'bmw']
>>> motorcycles.insert(1, 'jeep')
>>> print(motorcycles)
['tesla', 'jeep', 'honda', 'yamaha', 'suzuki', 'bmw']
>>> l1 = [1, 2, 3]
>>> l2 = [4, 5, 6]
>>> l1.extend(l2)
>>> print(l1)
[1, 2, 3, 4, 5, 6]
>>> l3 = [7, 8, 9, 10]
>>> print(l1 + l3)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>
```

Changing, Adding, and Removing Elements

- Removing elements from a list
  - Use the del statement
  - Use the pop() method
  - Removing an item by value

```
>>> motorcycles = ['honda', 'mazda', 'yamaha', 'suzuki', 'bmw']
>>> del motorcycles[0]
>>> print(motorcycles)
['mazda', 'yamaha', 'suzuki', 'bmw']
>>> motorcycles.pop()
'bmw'
>>> print(motorcycles)
['mazda', 'yamaha', 'suzuki']
>>> motorcycles.pop(1)
'yamaha'
>>> print(motorcycles)
['mazda', 'suzuki']
>>> motorcycles = ['honda', 'mazda', 'yamaha', 'suzuki', 'bmw']
>>> print(motorcycles)
['honda', 'mazda', 'yamaha', 'suzuki', 'bmw']
>>> motorcycles.remove('honda')
>>> print(motorcycles)
['mazda', 'yamaha', 'suzuki', 'bmw']
>>>
```

Other useful operators on lists

- The 'in' operator
- The + operator
- Get the index of an element
- Count some specific element in a list

```
>>> motorcycles = ['honda', 'mazda', 'suzuki', 'bmw', 'jeep']
>>> 'honda' in motorcycles
True
>>> 'tesla' in motorcycles
False
>>> motorcycles.index('bmw')
3
>>> motorcycles.count('jeep')
1
>>> motorcycles.append('jeep')
>>> print(motorcycles)
['honda', 'mazda', 'suzuki', 'bmw', 'jeep', 'jeep']
>>> motorcycles.count('jeep')
2
```

Organizing a list

- Sort a list
- Reverse order
- Finding the length of a list

```
>>> numbers = [8, 3, 6, 2, 1, 5]
>>> numbers.sort()
>>> print(numbers)
[1, 2, 3, 5, 6, 8]
>>> numbers.reverse()
>>> print(numbers)
[8, 6, 5, 3, 2, 1]
>>> len(numbers)
6
```

More resources : https://docs.python.org/3/tutorial/datastructures.html

Some useful operations on Strings

- Convert to upper case
- Convert to lower case
- Remove whitespaces

```
>>> str_test = "hello python"
>>> str_test.upper()
'HELLO PYTHON'
>>> print(str_test)
hello python
>>> str_test2 = "HELLO WORLD"
>>> str_test2.lower()
'hello world'
>>> str_whitespace = " hello world "
>>> str_whitespace.strip()
'hello world'
>>> str_whitespace
' hello world '
```

More resources : https://docs.python.org/3.7/library/string.html

Next Week

- Dictionary
- Control statement, if/else for loops, etc