CS3245

Information Retrieval

Python - NLTK

(Compiled by LIN Ziheng, updated by Abhinav Ramesh KASHYAP)

Python

INSTALLATION

Install Python on Windows

- Download the installer from https://www.python.org/downloads/windows/

- Run the installer
- Follow the instructions to install Python
- The default installed directory should be like `C:\Python(major)(minor)`

Install Python on Mac

- Most Mac machines come with Python installed: try typing `python3 --version`

- If you want to manually install (possibly a different version), download the installer from https://www.python.org/downloads/mac-osx/

Install Python on Linux

- If you are using Ubuntu, Python is installed by default, try typing `python3 --version`

- If you want to manually install (possibly a different version), download the source files from https://www.python.org/downloads/source/

- Unzip the file, cd to the unzipped directory, and run the normal "./configure → make → sudo make install" commands

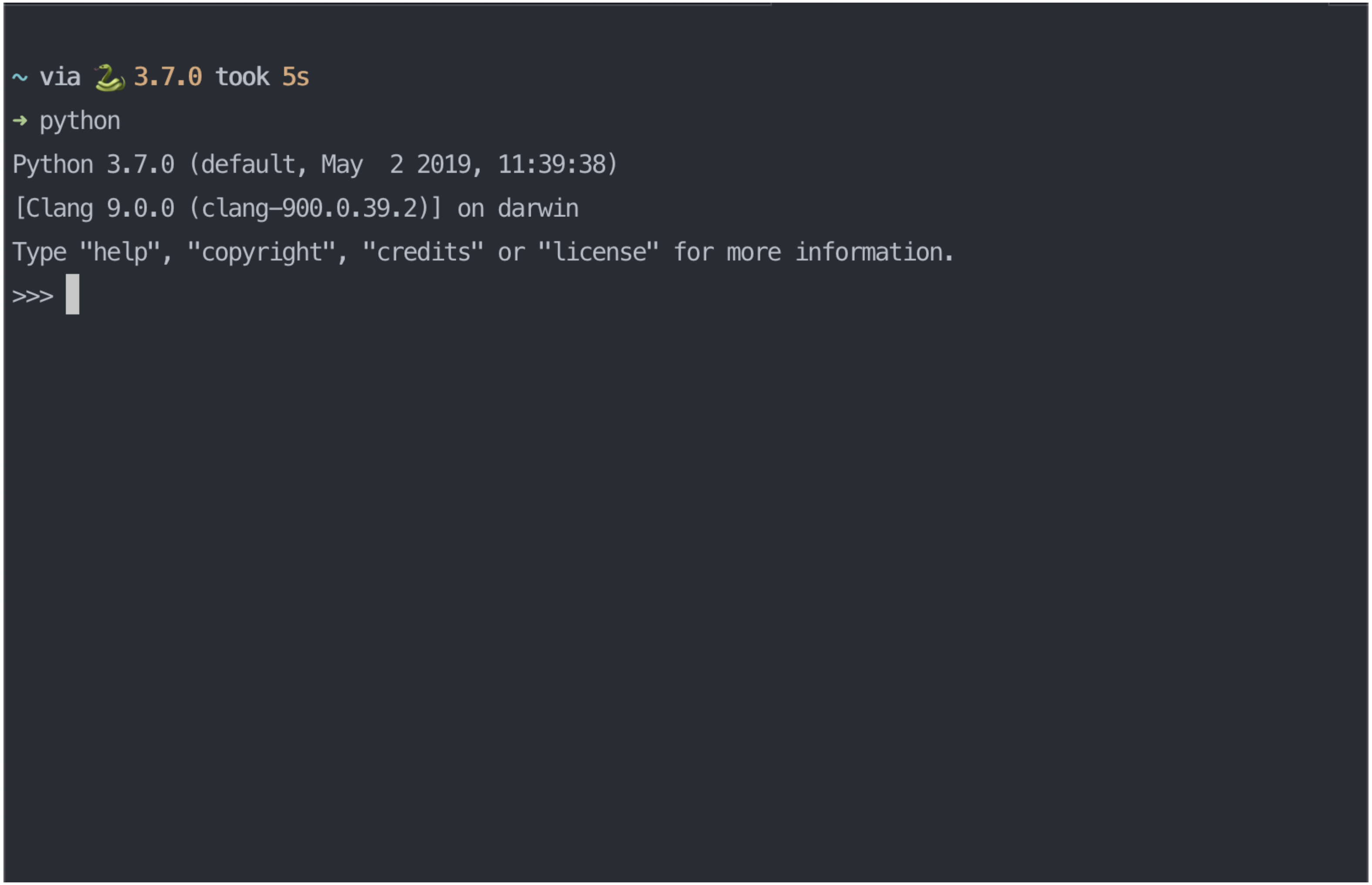
Python

CRASH COURSE IN PYTHON

Python Interpreter

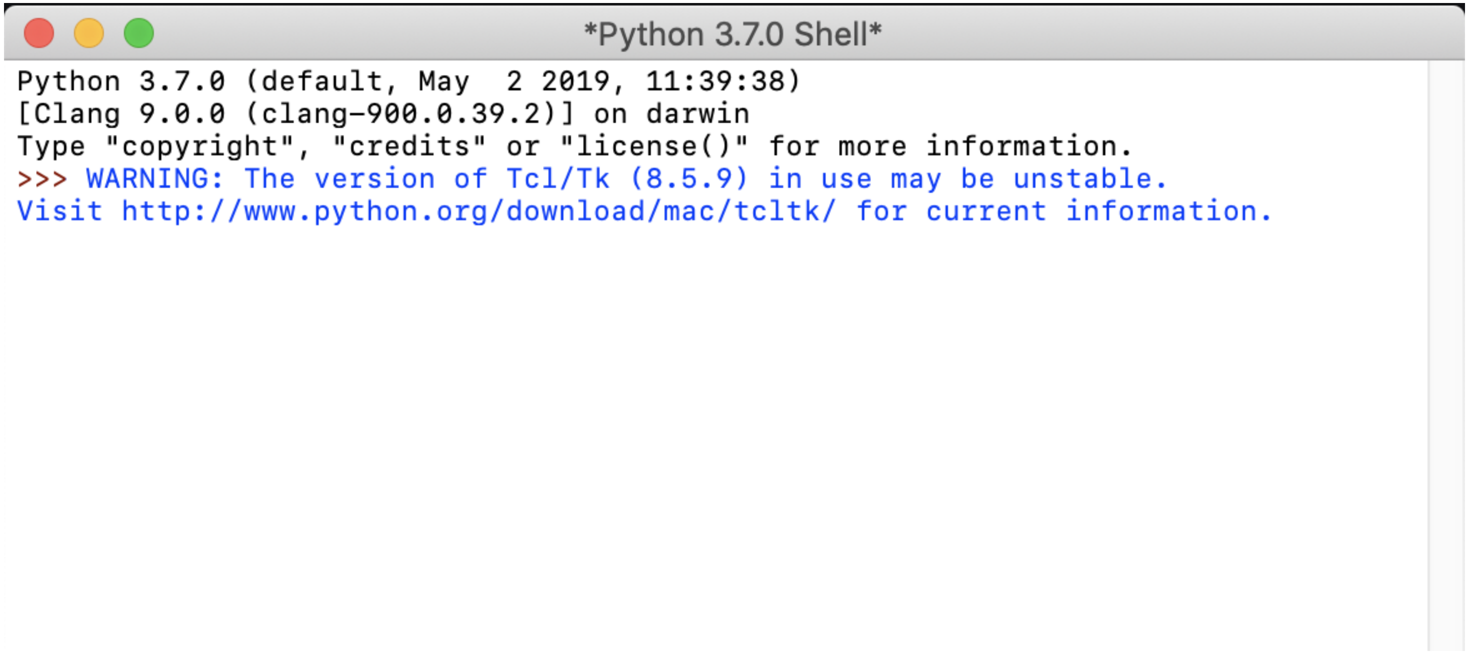
- Interactive interface to Python
  - On Windows: Start→All Programs→Python 3.7→Python (command line)
  - On Linux: type `python`
  - Python prompt: `>>>`
  - To exit:
    - On Windows: Ctrl-Z + <Enter>
    - On Linux: Ctrl-D
- Run python program on Linux:

```
% python filename.py
```



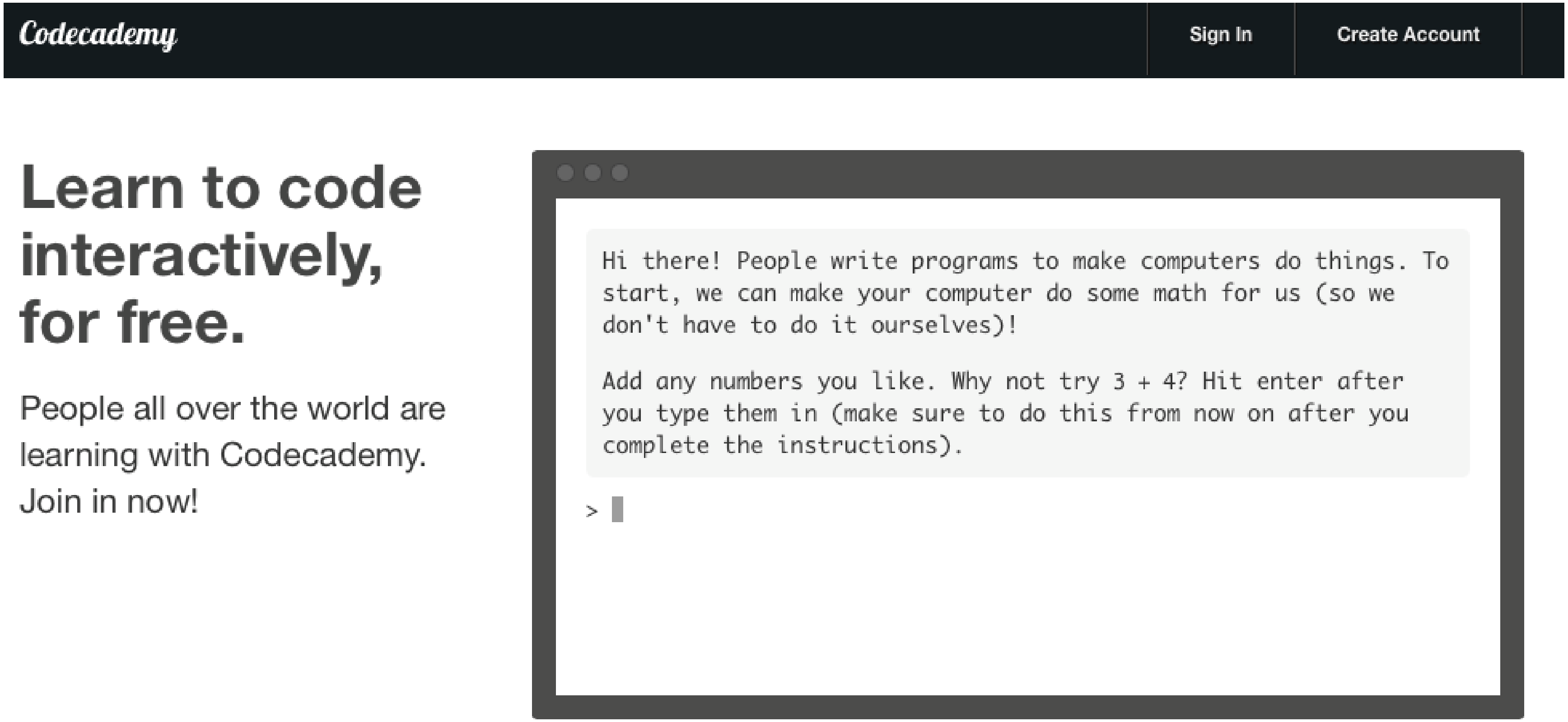
```
~ via 🐍 3.7.0 took 5s
➜ python
Python 3.7.0 (default, May  2 2019, 11:39:38)
[Clang 9.0.0 (clang-900.0.39.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

- IDLE: the Python IDE
  - On Windows: Start→All Programs→Python 3.7→IDLE (Python GUI)
  - On Linux and Mac: type `idle`



```
Python 3.7.0 (default, May  2 2019, 11:39:38)
[Clang 9.0.0 (clang-900.0.39.2)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.
```

Codecademy



Learn to code interactively, for free.

People all over the world are learning with Codecademy. Join in now!

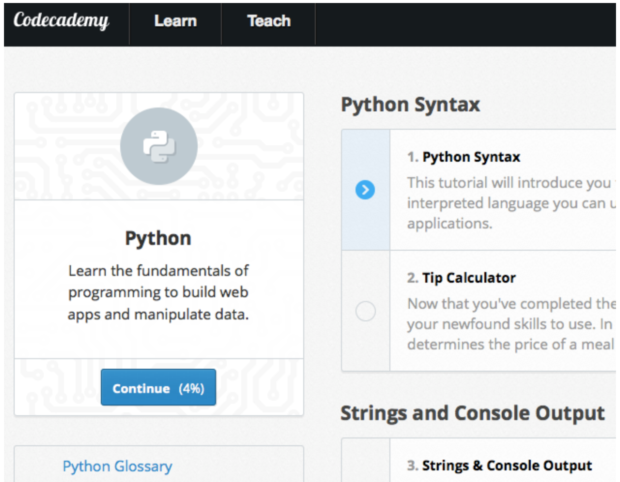
```
Hi there! People write programs to make computers do things. To
start, we can make your computer do some math for us (so we
don't have to do it ourselves)!

Add any numbers you like. Why not try 3 + 4? Hit enter after
you type them in (make sure to do this from now on after you
complete the instructions).

>
```

For today’s exercises

1. Sign up for Codecademy now
2. Pick the Learn Python 2 course

(Note that the Python 3 course is a pro feature so we will still refer to the Python 2 course instead for the relevant exercises. Just try to solve those exercises in Python 3.)

3. Jump to an appropriate exercise sequence



Python

Learn the fundamentals of programming to build web apps and manipulate data.

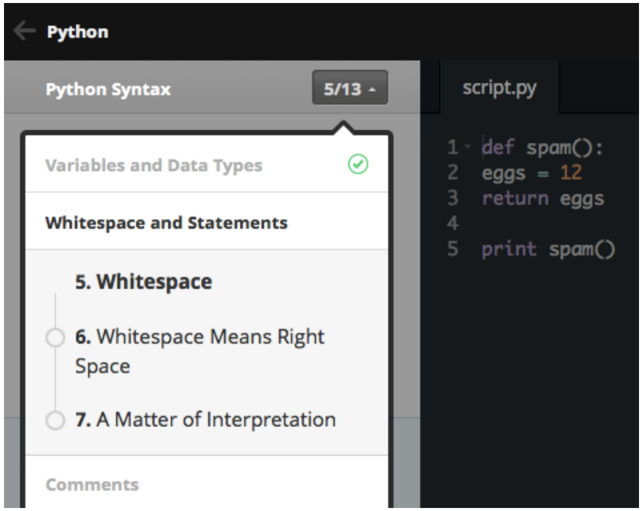
Python Syntax

1. **Python Syntax** This tutorial will introduce you interpreted language you can u applications.

2. **Tip Calculator** Now that you've completed the your newfound skills to use. In determines the price of a meal

Strings and Console Output

3. **Strings & Console Output**



Variables and Data Types ✓

Whitespace and Statements

5. **Whitespace**
6. Whitespace Means Right Space
7. A Matter of Interpretation

Comments

```
def spam():
eggs = 12
return eggs

print spam()
```

The Basics

- Numbers: integers, long integers, floating points, and complex numbers
- Strings:
  - Single quotes: `'hello!'`
  - Double quotes: `"what's your name?"`
  - Triple quotes: multi-line string

```
'''This is multi-line
string.
'''
```

  - Immutable: once created, cannot change
  - Concatenation: `'hi ' 'there'` same as `'hi ' + ‘there'` ***(Be careful)***

- Variables: `x = 1, y = 'NUS', x = y = z = 'SoC'`
- Identifier naming:
  - 1st char: a letter of the alphabet (upper or lowercase) or an underscore (‘_’)
  - The rest: letters (upper or lowercase), underscores (‘_’), or digits (0-9)
  - Case-sensitive: myname != myName
- Reserved words:
  - `and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while`

- Strongly object oriented: everything is an object, including numbers, string, functions
- Statements and semicolons:
  - Don't need to put (;) if you write one statement in one single line

```
i = 5
print(i)
```

  - Statements in one line are separated by (;)

```
i = 5; print(i)
```

  - Continuing a line with (\)

```
s = 'This is a string. \
This continues the string.'
```

- Indentation:
  - No braces { } to mark blocks of code
  - Leading whitespaces (spaces and tabs) are important
  - Statements in the same block have same indentation

```
i = 5
    print(i)
```

wrong

```
i = 5
print(i)
```

correct

  - Recommendation: consistently use a single tab or 2/4 spaces

- Comments:
  - Single line uses #
  - Multi-line uses “““…”””

```
# this is single line comment
""" this is multiple
line comment """
```

Operators and Expressions

- Operators:

```
+,  -,  *,  **,    /,  //,    %,
<<,    >>,    &,  |,  ^,  ~,
<,  >,  <=,    >=,    ==,    !=,
not,   and,   or
```

- Expression:

```
length = 5
breadth = 2
area = length * breadth
print 'Area is', area
print 'Perimeter is', 2 * (length + breadth)
```

Pretty print:

Try: `print(f”He is {5} years old”)`

`Works in python 3.7`

- Assignment uses =, comparison uses ==
- Multiple assignments: `x, y = 2, 'abc'`

Control Flow

- if…elif…else statement:

```
number = 23
guess = int(raw_input('Enter an integer : '))
if guess == number:
   print(‘Congratulations, you guessed it.’) # New block starts here
   print(“(but you do not win any prizes!)”) # New block ends here
elif guess < number:
   print(‘No, it is a little higher than that’) # Another block
   # You can do whatever you want in a block ...
else:
   print(‘No, it is a little lower than that’)
   # you must have guess > number to reach here
print 'Done'
```

- `while` statement:

```
number = 23
running = True
while running:
    guess = int(raw_input('Enter an integer : '))
    if guess == number:
        print(‘Congratulations, you guessed it.’)
        running = False # this causes the while loop to stop
    elif guess < number:
        print(‘No, it is a little higher than that.’)
    else:
        print(‘No, it is a little lower than that.’)
else:
    print(‘The while loop is over.’)
    # Do anything else you want to do here
print 'Done'
```

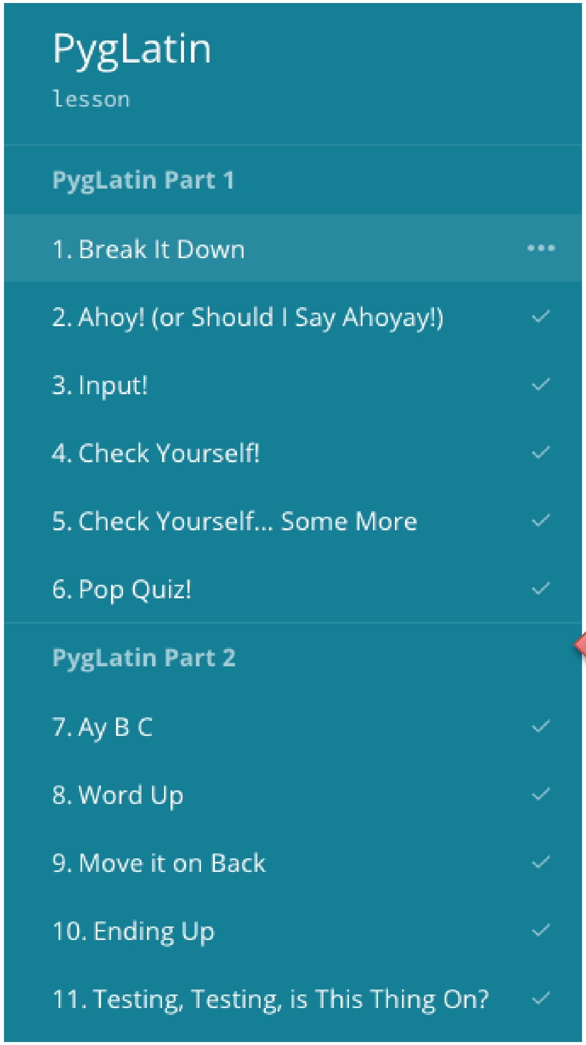
- `for` statement:

```
for i in range(1,5):
    print(i)
else:
    print('the for loop is over')
```

  - `range(1,5)` generates the list `[1,2,3,4]`
  - `range(a,b,s)` generates a list from a to b-1 with a step s
    - `list(range(1,10,2))` → `[1,3,5,7,9]`

- `break` statement:
  - break from current loop
- `continue` statement:
  - skip the rest of the current iteration and continue to the next iteration

Codecademy time!



PygLatin

lesson

PygLatin Part 1

1. Break It Down •••
2. Ahoy! (or Should I Say Ahoyay!) ✓
3. Input! ✓
4. Check Yourself! ✓
5. Check Yourself... Some More ✓
6. Pop Quiz! ✓

PygLatin Part 2

7. Ay B C ✓
8. Word Up ✓
9. Move it on Back ✓
10. Ending Up ✓
11. Testing, Testing, is This Thing On? ✓

Unit 3. PygLatin

- PygLatin Part 2 (Exercises 7-11)

Functions

- Defining a function with `def`:

```
def printMax(a, b):
    if a > b:
        print(f”{a} is maximum”)
    else:
        print(f”{a} is maximum”)
printMax(3, 4) # directly give literal values
x = 5
y = 7
printMax(x, y) # give variables as arguments
```

Function name and parameters

Function body

Outside the function

- Local variables: variables declared inside a function

```
>>> def func(x):
...     print(f”x is {x}”)
...     x = 2
...     print(f”Changed local x to {x}”)
...
>>> x = 50
>>> func(x)
x is 50
Changed local x to 2
>>> print(f”x is still {x}”)
x is still 50
```

- Use `global` to explicitly assign a value to a variable declared outside the function

```
>>> def func():
...     global x
...     print(f"x is {x}”)
...     x = 2
...     print(f”Changed global x to {x}”)
...
>>> x = 50
>>> func()
x is 50
Changed global x to 2
>>> print(f”Value of x is {x}”)
Value of x is 2
```

- Default argument value: make some parameters optional by providing default values

```
>>> def say(message, times = 1):
...     print(message * times)
...
>>> say('Hello')
Hello
>>> say('World', 5)
WorldWorldWorldWorldWorld
```

  - Note: only those at the end of the parameter list can be given default value
    - `def func(a,b=5)` is valid,
      `def func(a=5,b)` is not valid

- Keyword arguments: use names instead of positions to specify the arguments to the function

```
>>> def func(a, b=5, c=10):
...     print(f”a is {a}, b is {b}, c is {c}”)

>>> func(3, 7)
a is 3 and b is 7 and c is 10
>>> func(25, c=24)
a is 25 and b is 5 and c is 24
>>> func(c=50, a=100)
a is 100 and b is 5 and c is 50
```

- Use `return` to break out of a func and/or return a value

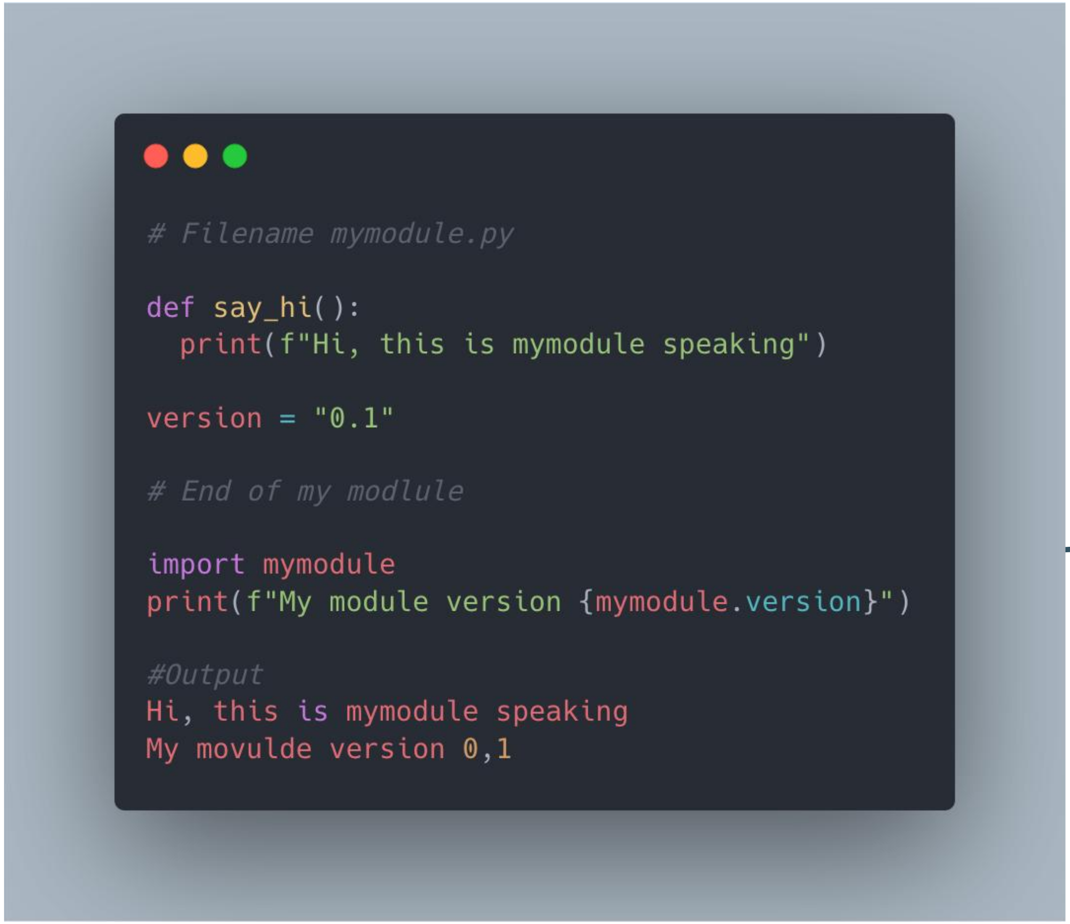
```
def maximum(x, y):
    if x > y:
        return x
    else:
        return y
print(maximum(2, 3))
```

Modules

- Module: a file containing all functions and variables that you have defined
- The module file should end with .py
- Use `import` to import a module:
  - Like Java *import* and C++ *include*
  - 3 formats:
    - `import somefile`
    - `from somefile import *`
    - `from somefile import className`
  - Import standard library and math library:

```
import sys
import math
```

- Make your own module:



```
# Filename mymodule.py

def say_hi():
  print(f"Hi, this is mymodule speaking")

version = "0.1"

# End of my modlule

import mymodule
print(f"My module version {mymodule.version}")

#Output
Hi, this is mymodule speaking
My movulde version 0,1
```

Save the module into mymodule.py

Import mymodule from the same dir

Data Structures

- Built-in data structures: list, tuple, dictionary
- List:
  - Specified by `[item0, item1, …]`
  - Stores a sequence of items
  - List is mutable: we can add, remove, change items
- List can store different types of items:
  - `[1, 'nus', [3, 'soc'], None, True]`

```
# This is my shopping list
shoplist = ['apple', 'mango', 'carrot', 'banana']
print(f‘I have, {len(shoplist)}, items to purchase.’)
print(‘These items are:’), # Notice the comma at end of the line
for item in shoplist:
    print item,
print 'I also have to buy rice.'
shoplist.append('rice')
print(f”My shopping list is now {shoplist}”)
print(f”I will sort my list now”)
shoplist.sort()
print(f”Sorted shopping list is {shoplist}”)
print(f”The first item I will buy is {shoplist[0]}”)
olditem = shoplist[0]
del shoplist[0]
print(f”I bought the {olditem}”)
print(f”My shopping list is now {shoplist}”)
```

- Tuple
  - Specified by `(item0, item1, …)`
  - Like lists except they are immutable: cannot be modified
  - Used when you can assume the collection of items will not change
- Tuple can store different types of items:
  - `(1, 'nus', [3, 'soc'], None, True, (1, 'a'))`
  - Empty tuple: `()`
  - Tuple with one item: `(1, )` not `(1)`

```
zoo = ('wolf', 'elephant', 'penguin')
print(f‘Number of animals in the zoo is {len(zoo)}’
new_zoo = ('monkey', 'dolphin', zoo)
print(f”Number of animals in the new zoo is {len(new_zoo)}”)
print(f‘All animals in new zoo are {new_zoo}’
print(f‘Animals brought from old zoo are {new_zoo[2]}’)
print(f‘Last animal brought from old zoo is new_zoo[2][2]’)

Output:
Number of animals in the zoo is 3
Number of animals in the new zoo is 3
All animals in new zoo are ('monkey', 'dolphin', ('wolf',
'elephant', 'penguin'))
Animals brought from old zoo are ('wolf', 'elephant', 'penguin')
Last animal brought from old zoo is penguin
```

- Tuples and the `print` statement: one of the most common usage of tuple, use % to format output

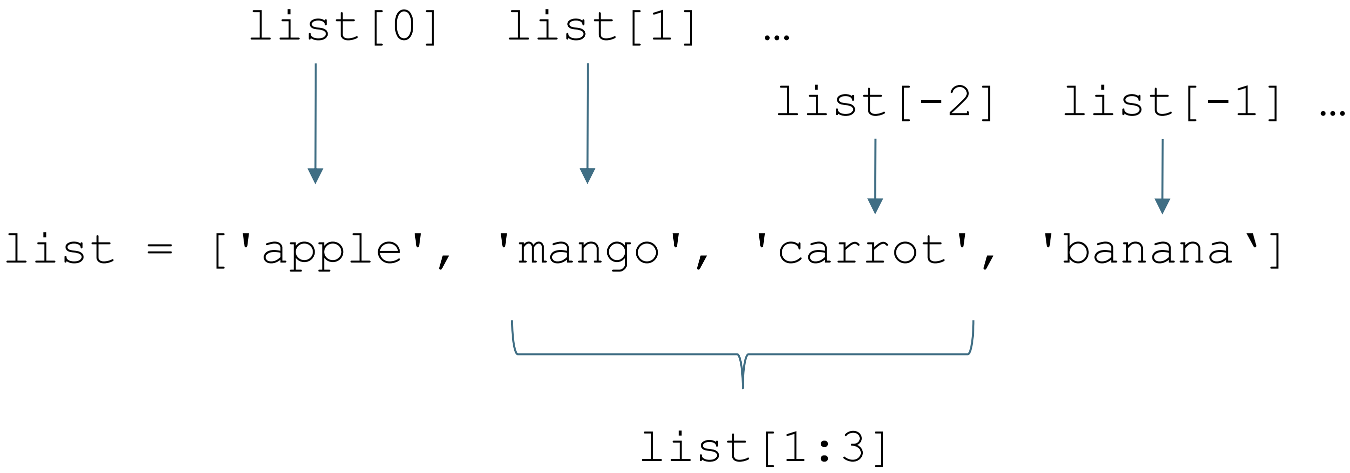
```
age = 22
name = 'Alex'
print(f’{name} is {age} years old’)
print(‘Why is {name} playing with that python?’)

Output:
Alex is 22 years old
Why is Alex playing with that python?
```

- Dictionary:
  - hash with key/value pairs
  - Keys must be unique
  - Specified by:
    - `{key0:value0, key1:value1, …}`

```
ab = { 'Alex' : 'alex@gmail.com',
    'Bob' : ‘bob@yahoo.com’}
print(“Alex's email is {ab[‘Alex’]}“)

# Adding a key/value pair
ab['Cindy'] = 'cindy@gmail.com'

# Deleting a key/value pair
del ab['Alex']

print(f‘There are {len(ab)} contacts in the address-book)

for name, address in ab.items():
    print(f‘Contact {name} at {address}’)
if 'Cindy' in ab:     # OR ab.has_key('Cindy')
    print(f“Cindy's email is {ab[‘Cindy’]}“)

Output:
Alex's email is alex@gmail.com
There are 2 contacts in the address-book
Contact Bob at bob@yahoo.com
Contact Cindy at cindy@gmail.com
Cindy's email is cindy@gmail.com
```

- Sequences:
  - Examples: list, tuple, string
- 2 main features of sequences:
  - Indexing: fetch a particular item
    - `[1,'a'][1], (1,'a')[1], 'hello'[1]`
  - Slicing: retrieve a slice of the sequence
    - `'hello'[1:4]  => 'ell'`
- Key difference:
  - Tuples and strings are immutable
  - Lists are mutable

- Indexing and slicing a sequence:



```
list[0]   list[1]   …
                       list[-2]   list[-1] …

list = ['apple', 'mango', 'carrot', 'banana‘]

                    list[1:3]
```

```
>>> a = "Singapore"
>>> a[2]
'n'
>>> a[:]
'Singapore'
>>> a[2:5]
'nga'
```

- Sequence operators:
  - `in`: boolean test whether an item is inside a sequence

    ```
    1 in [2, 'a', 1] → True
    'a' in 'abcd' → True
    ```

  - `+`: produces a new sequence by joining two

    ```
    (1, 2) + (3, 4) → (1, 2, 3, 4)
    'ab' + 'cd' → 'abcd'
    ```

  - `*`: produces a new sequence by repeating itself

    ```
    [1, 2] * 2 → [1, 2, 1, 2]
    'Hello' * 3 → 'HelloHelloHello'
    ```

- Sequence methods:
  - `len(s)`: return length of the sequence s
  - `min(s)` and `max(s)`: return the min and max value in s
  - `list(s)`: convert a sequence to a list

- List: + vs `extend()` vs `append()`
  - + creates a fresh list (new memory reference)
  - extend a list with another list
  - append a list with another item

```
>>> a = [1,2]
>>> b = [3,4]
>>> a + b
[1, 2, 3, 4]
>>> a + b
[1, 2, 3, 4]
>>> a.append([5,6])
>>> a
[1, 2, [5, 6]]
>>> a.append(7)
>>> a
[1, 2, [5, 6], 7]
>>> b.extend([5,6])
>>> b
[3, 4, 5, 6]
```

A new list

- More list methods:
  - `s.count(x)`: counts the occurences of an element in a list
  - `s.index(x)`: finds the first location of an element in a list
  - `s.remove(x)`: searches for and removes an element in a list
  - `s.sort()`: sorts a list
  - `s.reverse()`: reverses the order of a list

- References: when you bind a variable and an object, the variable only refers to the object and does not represent the object itself
- A subtle effect to take note:

```
list1 = ['a', 'b', 'c']
list2 = list1 # list2 points to the same list object
list3 = list1[:]    # list3 points to a new copy
```

- More string methods
  - `str1.startswith(str2)`: check whether str1 starts with str2

    `'Hello'.startswith('He')` → `True`
  - `str2 in str1`: check whether str1 contains str2

    `'ell' in 'Hello'` → `True`
  - `str1.find(str2)`: get the position of str2 in str1; -1 if not found

    `'Hello'.find('ell')` → `1`

- String ←→ list
  - `delimiter.join(list)`: join the items in list with delimiter

    ```
    '_'.join(['a', 'b', 'c']) → 'a_b_c'
    ```

  - `str.split(delimiter)`: split the str with delimiter into a list

    ```
    'a_b_c'.split('_') → ['a', 'b', 'c']
    ```

- More dictionary methods:
  - `a[k] = x`: sets a value in the dictionary
  - `a.has_key(k)`: tests for the presence of a keyword
  - `a.get(k, d)`: returns a default if a key is not found
  - `a.keys()`: returns a list of keys from a dictionary
  - `a.values()`: returns a list of values

Typing in Python

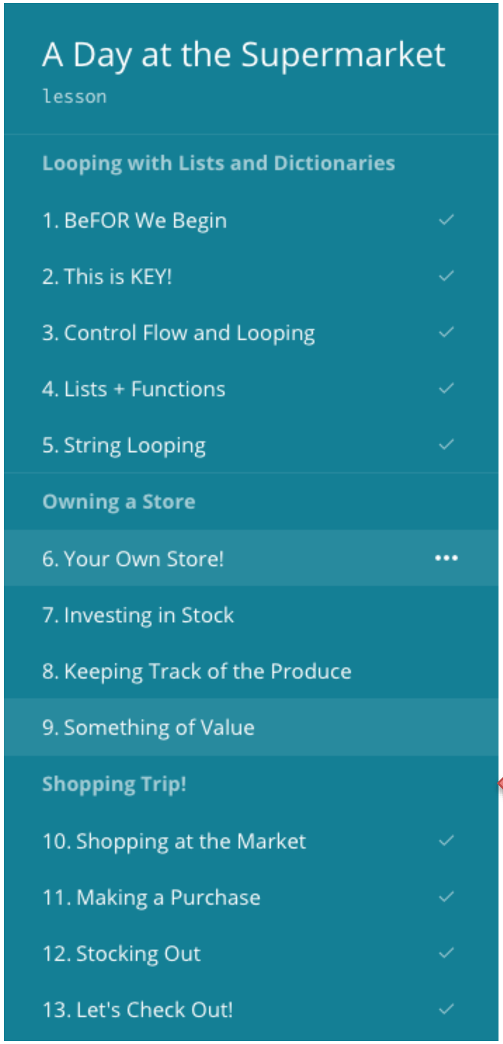
- Built-in types: str, bytes, list, tuple, set, dict, int, float, complex, bool
- Dynamic typing: determines the data types of variable bindings automatically

```
var = 2
var = 'hello'
```

- Strong typing: enforces the types of objects

```
    >>> print 'The answer is ' + 23
    Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
    TypeError: cannot concatenate 'str' and 'int'
objects
    >>> print 'The answer is ' + str(23)
    The answer is 23
```

Codecademy time!



A Day at the Supermarket

lesson

Looping with Lists and Dictionaries

1. BeFOR We Begin ✓
2. This is KEY! ✓
3. Control Flow and Looping ✓
4. Lists + Functions ✓
5. String Looping ✓

Owning a Store

6. Your Own Store! •••
7. Investing in Stock
8. Keeping Track of the Produce
9. Something of Value

Shopping Trip!

10. Shopping at the Market ✓
11. Making a Purchase ✓
12. Stocking Out ✓
13. Let's Check Out! ✓

Unit 5. A Day at the Supermarket

- Shopping Trip! (Exercises 10-13)

Object-oriented Programming

- Class: a data type
- Object: an instance of the class
- Fields: variables that belong to an object or class
  - Two types: instance variables and class variables
- Methods: functions that belong to a class
- Fields and methods are referred to as the attributes of that class

- The `self`:
  - The first parameter of a class method is the `self` (similar to *self* in C++ and *this* in Java)
  - But you don't need to specify `self` when calling the method

- Creating a simple class:

```
class Person:
    def sayHi(self):
        print(‘Hi there!’)

p = Person()
print(p)
p.sayHi()

Output:
<__main__.Person instance at
0xf6fcb18c>
Hi there!
```

p.sayHi() is internally transformed into Person.sayHi(p)

This tells us we have an instance of the Person class in the __main__ module and its address in the memory

- The `__init__` method:
  - Is run as soon as an object is instantiated
  - Analogous to a constructor in C++ and Java

```
class Person:
    def __init__(self, name):
        self.name = name
    def sayHi(self):
        print(f‘Hello, my name {self.name}’)

p = Person('Jack')
p.sayHi()

Output:
Hello, my name is Jack
```

- Class and object variables:
  - Class variable: accessed by all objects of the class
    - Changes will be seen by all objects
  - Object variables: owned by each individual object of the class
    - Not shared by other objects of the same class

```
class Person:
    '''Represents a person.'''
    population = 0
    def __init__(self, name):
        '''Initializes the person's data.'''
        self.name = name
        print(f‘Initializing {self.name}’)
        # When this person is created, he/she
        # adds to the population
        Person.population += 1
    def __del__(self):
        '''I am dying.'''
        print(’{self.name} says bye.’)
        Person.population -= 1
        if Person.population == 0:
            print(‘I am the last one.’)
        else:
            print(f‘There are still {Person.population} people left)
    def sayHi(self):
        '''Greeting by the person.'''
        print(f‘Hi, my name is {self.name}.’)
    def howMany(self):
        '''Prints the current population.'''
        if Person.population == 1:
            print(‘I am the only person here.’)
        else:
            print(‘We have {Person.population} persons here.’)
```

population is a class variable, so is referred as Person.population

name is an object variable, so is referred as self.name

__del__ is called when the object is going to die

```
alex = Person('Alex')
alex.sayHi()
alex.howMany()
bob = Person('Bob')
bob.sayHi()
bob.howMany()
alex.sayHi()
alex.howMany()
```

```
Output:
(Initializing Alex)
Hi, my name is Alex.
I am the only person here.
(Initializing Bob)
Hi, my name is Bob.
We have 2 persons here.
Hi, my name is Alex.
We have 2 persons here.
Bob says bye.
There are still 1 people left.
Alex says bye.
I am the last one.
```

- Inheritance:
  - Implement a type and subtype relationship between classes
  - Reuse of code
  - Multiple inheritance
  - Declared by:

```
class DerivedClass(Base1, Base2, …)
```

```
class SchoolMember:
    '''Represents any school member.'''
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print(f”Initialised SchoolMember {self.name}”)
    def tell(self):
        '''Tell my details.'''
        print(f’Name: {self.name} Age: {self.age}’ )
class Teacher(SchoolMember):
    '''Represents a teacher.'''
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print(f“Initialised Teacher: {self.name}”)
    def tell(self):
        SchoolMember.tell(self)
        print(f‘Salary: {self.salary}’)
class Student(SchoolMember):
    '''Represents a student.'''
    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
        self.marks = marks
        print(f’Initialised Student: {self.name}’)
    def tell(self):
        SchoolMember.tell(self)
        print(f’Marks: {self.marks}’)
```

Call base class __init__

Call base class method

I/O

- Files:
  - Create an object of the `file` class to use the `read`, `readline`, or `write` method
- Write to a file:

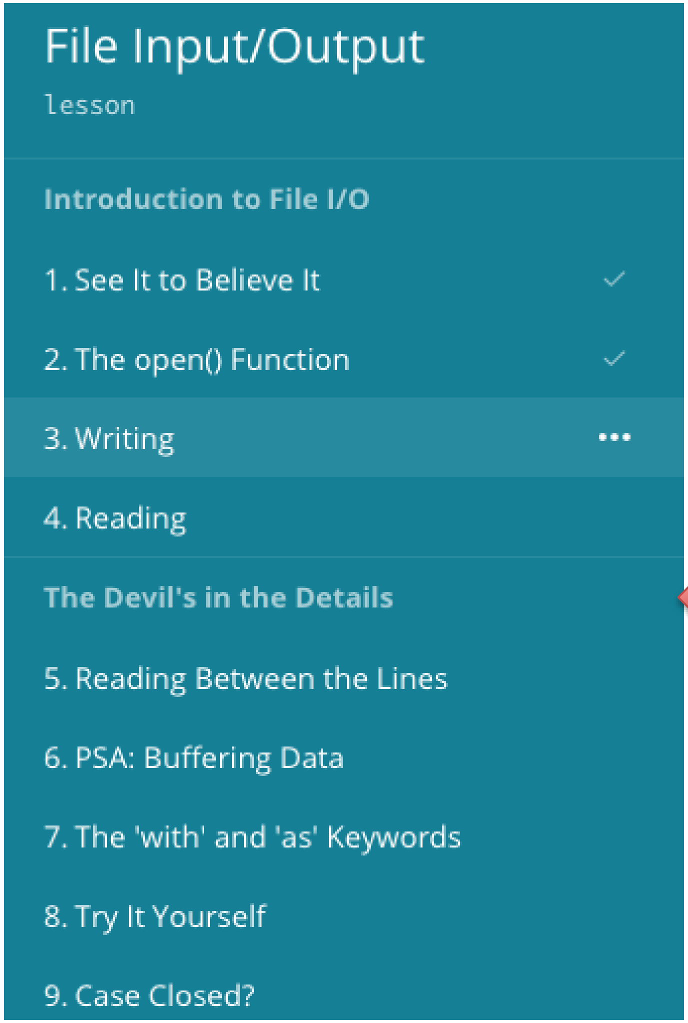
```
f = file('file.txt', 'w')
f.write(str)
f.close()
```

'w' for write
'r' for read
'a' for append

- Read from a file:

```
f = file('file.txt')
for line in f:
      print line
f.close()
```

Codecademy time!



File Input/Output

lesson

Introduction to File I/O

1. See It to Believe It ✓
2. The open() Function ✓
3. Writing •••
4. Reading

The Devil's in the Details

5. Reading Between the Lines
6. PSA: Buffering Data
7. The 'with' and 'as' Keywords
8. Try It Yourself
9. Case Closed?

Unit 12. File Input/Output

- The Devil’s in the Details (Exercises 5-9)

- `Pickle`:
  - Use the `pickle` module to store any object to a file so that you can get it back later intact → storing object persistently
  - Another module `cPickle` is written in C, and is upto 1000 times faster

```
import cPickle as p
#import pickle as p

shoplistfile = 'shoplist.data'
shoplist = ['apple', 'mango', 'carrot']

# Write to the file
f = file(shoplistfile, 'w')
p.dump(shoplist, f) # dump the object to a file
f.close()
del shoplist # remove the shoplist

# Read back from the storage
f = file(shoplistfile)
storedlist = p.load(f)
print storedlist
```

Exceptions

- Errors are objects
  - More specific kinds of errors are subclasses of the general Error class
- Catch errors:

```
try … except …
try … except … else …
try … except … else … finally …
```

- Raise errors:

```
raise …
```

```
while True:
    try:
        x = int(raw_input("Please enter a number: "))
        break
    except ValueError:
        print "That was not a valid number. Try again..."
```

Standard Library

- The `sys` module:
  - Contains system specific functionality
  - Use: `import sys`
  - `sys.argv`: list of arguments
  - `sys.exit()`
  - `sys.version`, `sys.version_info`: Python version information
  - `sys.stdin`, `sys.stdout`, `sys.stderr`
  - …

- The `os` module:
  - Generic operating system functionality
  - Important if you want to make your program platform-independent
  - Use: `import os`
  - `os.sep`: windows → '\\', linux → '/'
  - `os.name`: windows → 'nt', linux → 'posix'
  - `os.getcwd()`: get current working directory
  - `os.getenv()`, `os.putenv()`: get and set environment variables
  - `os.listdir()`: get names of all files in the specified directory
  - `os.remove()`: delete a file
  - `os.system()`: run a shell command
  - `os.linesep`: windows → '\r\n', linux → '\n', mac → '\r'
  - `os.path.split()`, `os.path.isfile()`, `os.path.isdir()`, …

More Python

- Special methods:
  - Used to mimic certain behavior
    - E.g.: to use indexing `x[key]` for your class, you implement the `__getitem__()` method
  - `__init__(self, …)`: called to instantiate an object
  - `__del__(self)`: called just before the object is destroyed
  - `__str__(self)`: called when we `print` the object or use `str()`
  - `__lt__(self, other)`: called when less than (<) is used
  - `__getitem__(self, key)`: called when `x[key]` is used
  - `__len__(self)`: called when `len()` is used

- Random numbers:
  - Print a random number in [0,1):

    ```
    import random
    print random.random()
    ```

  - `randrange(a, b)`: chooses an integer in the range [a, b)
  - `uniform(a, b)`: chooses a floating point number in the range [a, b)
  - `normalvariate(mean, sdev)`: samples the normal (Gaussian) distribution

- List comprehension:
  - Derive a new list from existing lists
  - Similar to the list comprehension in Haskell
  - Python programmers use list comprehension extensively
  - Syntax: `[expression for name in list]`

```
>>> [2*i for i in [2,3,4]]
[4, 6, 8]
>>>
>>> [n * 3 for (x, n) in [('a', 1), ('b', 2), ('c', 3)]]
[3, 6, 9]
>>>
```

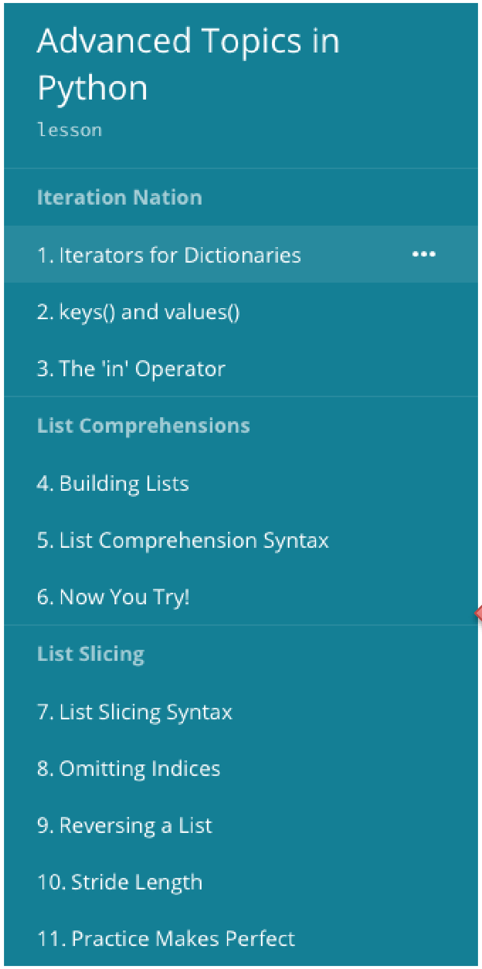
- Filtered list comprehension:
  - Use filter condition
  - `[expression for name in list if filter]`

```
>>> [2*i for i in [2,3,4] if i > 2]
[6, 8]
```

- More examples:

```
>>> [x*y for x in [1,2,3,4] for y in [3,5,7,9]]
[3, 5, 7, 9, 6, 10, 14, 18, 9, 15, 21, 27, 12, 20, 28, 36]
>>>
>>> [(x,y) for x in [1,3,5] for y in [2,4,6] if x < y]
[(1, 2), (1, 4), (1, 6), (3, 4), (3, 6), (5, 6)]
>>> [n * 2 for n in [m + 1 for m in [3,2,4]]]
[8, 6, 10]
>>>
```

Codecademy time!



Advanced Topics in Python

lesson

Iteration Nation

1. Iterators for Dictionaries •••
2. keys() and values()
3. The 'in' Operator

List Comprehensions

4. Building Lists
5. List Comprehension Syntax
6. Now You Try!

List Slicing

7. List Slicing Syntax
8. Omitting Indices
9. Reversing a List
10. Stride Length
11. Practice Makes Perfect

- Unit 10. Advanced Topics in Python
  - Iteration Nation (Exercises 1-3)
  - List Comprehensions (Exercises 4-6)
  - List Slicing (Exercises 7-11)

- Aggregating function arguments:
  - You can use * or ** to aggregate arguments in to a tuple or dictionary

```
def fun(a, *args):
    print(a)
    print(args)

fun(1, 3, 'a', True)

Output:
1
(3, 'a', True)
```

```
def fun(a, **args):
    print(a)
    print(args)

fun(1, b=3, c='a', d=True)

Output:
1
{'c': 'a', 'b': 3, 'd': True}
```

- Lambda forms:
  - Create anonymous functions at runtime
  - Powerful when used with `filter()`, `map()`, `reduce()`

```
>>> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>>
>>> print(list(filter(lambda x: x % 3 == 0, foo)))
[18, 9, 24, 12, 27]
>>>
>>> print(list(map(lambda x: x * 2 + 10, foo)))
[14, 46, 28, 54, 44, 58, 26, 34, 64]
>>>
>>> print(list(reduce(lambda x, y: x + y, foo)))
139
```

- The `exec` and `eval` statements:
  - `exec`: execute Python statements stored in a string or file
  - `eval`: evaluate valid Python expressions stored in a string

```
>>> exec('a = "Hi " + "there!"; print a')
Hi there!
>>> eval('2**3')
8
```

- The `repr()` and backticks (`) statement:
  - Return a printable representation of the object

```
>>> list = [1,'a',True,[33]]
>>> repr(list)
"[1, 'a', True, [33]]"
>>> `list`
"[1, 'a', True, [33]]"
```

NLTK

NATURAL LANGUAGE TOOLKIT (NLTK)

A Quick Look at NLTK

- Installing NLTK:
  - General instructions: https://www.nltk.org/install.html
  - On SoC Compute Cluster:
    - `pip install --user -U nltk`

- Installing NLTK data
  - Enter the Python command line (e.g., by typing python3)

    ```
    import nltk
    nltk.download()
    ```

  - Download the book collection.
  - You can then load some texts of several books by using:

    ```
    from nltk.book import *
    ```

  - Try typing `text1` and `text2`

- Searching the text
  - Show a concordance view of a word with its contexts:

    ```
    text1.concordance("monstrous")
    ```

    which shows contexts such as *the* ____ *pictures* and *the* ____ *size*
  - Show words that appear in the similar context:

    ```
    text1.similar("monstrous")
    ```
  - Examine the contexts that shared by two or more words:

    ```
    text2.common_contexts(["monstrous",
    "very"])
    ```

- Counting vocabulary
  - Count length of a text in terms of words and punctuations:

```
len(text3) → 44764
```

  - Count the vocabulary size of a text:

```
len(set(text3)) → 2789
```

  - Let’s measure the lexical richness of a text:

```
      float(len(text3)) / len(set(text3)) →
16.050197203298673
```

  - Count word occurrence:

```
text3.count("smote")
```

- Create lists from text items:

```
[w for w in set(text1) if len(w) > 15]
[len(w) for w in text1]
[w.upper() for w in text1]
```

Python - NLTK

ADDITIONAL PRACTICE

Practice 1 - Python

- Write a program to count word occurrences in a file.
  - Convert all words to lowercase
  - Excludes numbers and punctuation
  - Print the words and word counts by descending frequency
  - Reads the file name as the only argument

```
% python count_words.py filename.txt
```

Can be (comprehensibly) done on one line!

Practice 2 - Python

- The “paper, scissors, stone” game: write a program to play “paper, scissors, stone” with the computer
  - User chooses how many points are required for a win
  - User keys in one of the three selections: (p)aper, (s)cissors, or s(t)one
  - Computer randomly generates one selection

```
Welcome to Paper, Scissors, Stone!

How many points are required for a win? 3

Choose (p)aper, (s)cissors, or s(t)one? t
Human: stone          Computer: paper      Computer wins!
Score: Human 0   Computer 1

Choose (p)aper, (s)cissors, or s(t)one? t
Human: stone          Computer: scissors   Human wins!
Score: Human 1   Computer 1

Choose (p)aper, (s)cissors, or s(t)one? p
Human: paper          Computer: paper      A draw
Score: Human 1   Computer 1

Choose (p)aper, (s)cissors, or s(t)one? s
Human: scissors       Computer: paper      Human wins!
Score: Human 2   Computer 1

Choose (p)aper, (s)cissors, or s(t)one? t
Human: stone          Computer: scissors   Human wins!
Final Score: Human 3   Computer 1
```

Practice 3 - NLTK

- Write expressions for finding all words in text6 that meet the following conditions. The result should be in the form of a list of words: ['word1', 'word2', ...].
  - Ending in *ize*
  - Containing the letter *z*
  - Containing the sequence of letters *pt*
  - All lowercase letters except for an initial capital (i.e., titlecase)

Practice 4 - NLTK

- Take a text in the nltk set (say `text1`)
- Make a function that prints the 5 highest frequency stem (using Porter's stemmer) and the 5 highest frequency words in the corpus.

- `FreqDist()` may come in handy.