Python Bootcamp

Section 3, Lecture 10 Data Types

int
float
str
list (ordered objects) — mutable
dict Dictionary, Unordered Key:value pairs k1:v1, k2:v2
tup Tuples, Ordered, unchangeable, list
set Unordered collection of objects
bool

S3, L11 Arithmetic

Modulo % remainder of division

S3, L13 Var Assign

type(var) function returns data type of var

S3 L14 Strings

indexing strings. 1st letter is 0,
→ last letter has index -1

slice (or substring) is indexed as
[start:stop:step]
len(string) length

S3, L15

When slicing, can use step of -1 to reverse the string

S3, L16

String concatenation is '+' plus sign.

Use * to make strings repeat

2 * "hi" ⇒ "hi hi"

Methods on strings available in Jupyter after typing period '.' after string and hitting tab.

Also methods for ints exist.

Methods:

upper(), lower(), capitalize(), split(",")

S3, L18 Print formatting strings

print(string {n} {n2} {n3}.format(s2, s2, s2))

format for float

print("value" {val: width.precf}.format(val))

f string formatting

print(f'{var1} strings {var2} strings')

53, L20 Lists []

lists can be changed & can be indexed & sliced can contain mixed data types

lists have . (dot) methods

.append() .pop(i) .sort()

None is a null item (no value)

.reverse()

nested list eg. my_list[n][n]

53, L22 Dictionaries {}

Dictionaries can't be sorted, mutable
can be nested dict[k1][k2]
can have lists inside dict[k1][2]

53, L24 Tuple ()

Immutable can be indexed & sliced

S3, L25 Sets, set()

Unordered collection of UNIQUE elements

set() is a casting function. can cast a list [] into a set

S3, L27 Files

In Jupyter Notebook can create text file such

```
%%% filename.txt
Line 1
Line 2
⋮
```

pwd prints working directory

can use methods .open(), .read(), .seek(), .close() on files

to avoid having to open/close can use copy using the "with" keyword

e.g.
```
with open('myfile.txt') as file2:
    contents = file1.read()
```

S3, L29 Test 1

Numbers: Numerical values like integers of floats
Strings: Words made of characters, a character array
Lists: a collection of elements
Tuples: Immutable unordered lists of elements
Dictionaries: key: value pairs mutable

S4, L31 Comparison

Like C == & !=

S4, L32 Chaining Comparison

a<b>c works!
a<b and b>c

S5, L3 For Loops

for placeholder in iterable:
    block

Common syntax in for loop when placeholder not needed is use underscore '_'

Tuple unpacking: assign multiple comma-separated placeholders that each correspond to elements in the tuple

SS, L34 For loops Cnt'd

Tuple unpacking also works with dictionaries. When only Dictionary name is used (for item in dict:) item iterates through the keys. Use dict.items() to get the key,value pair. Use dict.values() to get just the values

SS, L35 While loops

While loops can have an else

break	out of loop
continue	go back to top of enclosing loop
pass	do nothing

SS, L36 Useful Operators

for num in range(start, end+1, step)
enumerate() steps through objects in list
zip() zips together lists & returns tuples
    ties through shortest list ignores res
val in list
min() max()

from random import shuffle { random is a library
loads function shuffle (in place) from library random

input('text prompt')
accepts strings
use casting functions int() or float()

55, L37 List Comprehension

List comprehension is a flattened out for loop
result = [item for item in list]
can use range( ) instead of list
can add if condition like ...
result = [x*2 for x in range(0,21) if x%2==0]
can do if else in list comprehension
result = [x if x%2==0 else 'Odd' for x in range(0,11)]
but is less readable
can nest for loops in list comprehension
result = [x+y for x in List1 for y in List2]

S6, L40 Methods & Functions

for lists exist .append() & .pop()
for other built-in methods visit docs.python.org/3

S6, L41 Functions

```
def name(parameter1, p2, ...):
    '''
    Docstring explains function
    '''
    block
```

this will pop up with <shift><tab> key

Pig Latin if starts with vowel add ay else move 1st letter to end & append ay

S6, L43 *args & *kwargs

Creates a tuple in func { *args arbitrary # of arguments
*name could be any name but args by convention

** kwargs just like *args but for dictionary pairs

can do both *args & **kwargs
but must be in that order.

S6, L49 Lambda Expressions

map(func, args)
e.g., for i in map(func, args):
        print(i)

filter(func, args) func must be bool

lambda x: expression, args

S6, L50 Statements & Scope

can define globals in funcs

global var

S7, Milestone Project

Completed. Source in Spyder & Notebook

58, L60 OOP Attributes of Class

```
class Sample():
    pass
my_sample = Sample()
```

replace with

e.g. class Dog(): ← class

```
    def __init__(self, breed)
        self.breed = breed
```

attribute

invoke with...

```
my_dog = Dog(breed = 'Lab')
```

58, L61 Class, Attrib, Method

Class object attribute goes between class statement and def instance definition. its the same for all instances of the class.

Method declared just like a function inside a class. use self. do use local variables in the class

58, L62 OOP Inheritance & Polymorphism

Inheritance, can create class with argument naming a previously created class. New class inherits all methods of class referenced in argument

Polymorphism provides a way to access methods in the class instance across various classes. Can iterate or call similar shared methods for various class instances.

Base classes can have abstract methods that are not implemented. It might not be intended to have instances of a base class. It is used to provide a structure & data to sub-classes. Abstract methods are just stubs that are expected to be over-written by subclasses that use the base class. Abstract methods should be written to "raise" exceptions when called.

S8, L63 Special Magic / Dunder Methods

To use built in functions, must define corresponding type inside of class.
e.g., def __str__(self):
    return <some string>

can delete variables with del keyword
e.g. del <var> __del__(self) can be defined in a class to perform an action on delete

S10, L71 Errors & Exceptions

```
try:
    {block}
except:
    {block}
else:
    {block}
finally:
    {block}
```

S10, L74 Pylint

pylint myscript.py

checks against convention

S10, L75 Unittest

```
import unittest
import cap

class TestCap(unittest.TestCase):
    def test_something_in_cap(self):
        text = 'test sample string capitalized'
        result = cap.cap_text(text)
        self.assertEqual(result, 'Python')
```

function in cap.py module tested

S11 Milestone Project 2

Blackjack game.
Completed successfully.
Runs w/o errors.
Paid special attention to plint
Ran pylint with score of 10.0.

S12. L81 Decorators

Decorators are a way to add functionality to functions, without altering original function

```
@some_decorator
def simple_func():
    # simple stuff
    return something
```

can copy functions to variable that become a function (not just pointer to function)
eg. copy_func = simple_func

This concept is used for decorators it's a wrapper for the original function.

S12, L82 Decorators Homework

Flask & Django

S13, L83 Python Generators

Use yield statement to return a value & suspend after the yield.

S14, Final Capstone Project.

1) Short project: Compute n digits precision of π. Wrote code an computed 10,000,000 digits precision.

2) SwimBoard scoreboard for swimming with current & prior race displays.

S15, L87 Collections Module

```
from collections import Counter
l = [1, 1, 3, 4, 3, 6, 7]
Counter(l)  counts occurences in l
```

There are comon patterns available on Counter objects. They are invoked like methods, eg.

```
Counter(l).clear()      #reset all counts
          .most_common()
sum(Counter(l).values())
```

etc.

S15, L88 defaultdict

```
from collections imprt defaultdict
```

defaultdict doesn't return error when referencing an item that isn't in the dict. Instead it initializes it

S15, L89 OrderedDict

keeps track of order items are added to dict. Ordered dicts that have the same data in different order are not equal

S15, L90 namedtuple

from collections import namedtuple

Dog = namedtuple('Dog', 'age breed name')
like creating a class

Sam = Dog(age = 2, breed = 'Lab', name = 'Sammy')
Sam.age gives 2

S15, L91 datetime module

import datetime
datetime.time(5h, 25mm, 1ss, ms)
and even timezone

today = datetime.date.today()

S15, L92 pdb Debugger

pdb.set_trace() { Turns on interactive debugger

Use 'q' to quit pdb

S15, L93: timeit: timing code

import timeit

create a string 0 - 99 with dash in between

note

eg.1.

```
timeit.timeit("-".join(str(n) for n in range(100)), number=10000)
```

number = repeat count

eg.2.

```
timeit.timeit("-".join(map(str, range(100))), number=10000)
```

btw, map is much faster

built in magic %

```
%timeit "-".join([str(n) for n in range(100)])
```

or

```
%timeit "-".join(map(str, range(100)))
```

S15, L94 Regular Expressions

import re

re.<method>( )

{there's a bunch

look it up.

S15, L95 StringIO

import StringIO { Implements file-like object in memory

```
f = StringIO.StringIO('some string')
f.read()    etc.
```

cStringIO is a C implementation of StringIO which is much faster.

S15, L96 FAQs

S16, L97 Advanced Numbers

hex() gives hex from dec
pow(num, exp[, mod])
abs() round(num, precision)
bin()

S16, L98 Advanced Strings

```
s = 'hello world'
s.capitalize(), s.upper(), s.lower()
s.count('o'), s.find('o')
s.center(<field size>, <fillchar>)
```

s.expandtabs() s.isalnum() } T or F
s.isalpha() s.islower() s.istitle()
s.isupper() s.endswith(<string>)
s.split() s.partition(<str>) ⇒ head, sep, tail

S16, L99 Advanced Sets

```
s = set()
s.add(1)
s.add(2)
s ⇒ {1, 2}
```

Methods on sets

```
s.clear()  s.copy()  s.difference(<set>)
s.difference_update(<set2>)   { removes elements in s that are in set2
s.discard(val)
s.intersection(s2)
s.intersection_update(s2)   { updates s with intersection of s & s2
s.isdisjoint(s2)   T if no intersection of s & s2
s.issubset(s2)   T/F
s.issuperset(s2)   T/F
s.symmetric_difference(s2)
s. ———— " ———— _update(s2)
s.union(s2)
s.update(s2)
```

S16, L100 Advanced Dictionaries

Dictionary Comprehension

```
{x: x**2 for x in range(10)}
```

Iteration over dict

```
for k in d.itervalues():
or... for k in d.iterkeys():
```

```
d.viewitems()
d.viewkeys()
d.viewvalues()
```

S16, L101 Advanced Lists

```
l.append()      , l.count()
L.extend()      , L.index()
L.insert(index, object)
l.pop()  or  L.pop(index)
L.remove(val)  { 1st instance
l.reverse()    } in place
l.sort()       }
```

S16, L102 Advanced Python Objects

Test

S17, L106 ipyWidgets

from ipywidgets import interact, interactive, fixed
import ... as widgets

Use decorator for interact

@interact(x=True, y=1.0)
def g(x, y):
    return (x, y)

or y = fixed(1.0)

interact(func, x=widgets.IntSlider(val, min, max, step))
or can do value=v.., min=?...
or
interact(func, x=(min, max, step))

Drop down menu use list not tuple
interact(func, x=['a', 'b', ...])
make dictionary to return value for dropdown
interact(func, x={'a': v1, 'b': v2, ...})
also
interactive

S 17, L 107 ipywidgets

display widget etc

S.17, L 108 list of widgets

This is a comprehensive list includes range slider, progress bar etc

S.17, L 109 Layout & Styling

lots of options

S 17, L 110 Example

Lorenz diff eq.