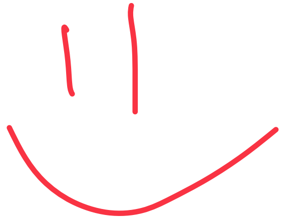
Algorithms and Data Structures for Data Science
Functions and Objects

CS 277
Brad Solomon

January 23, 2024

Major technical difficulties!

Annotation is incomplete.



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

CS 277 should be low-stress medium workload

If you are struggling to complete assignments, ask for help!

1. Attend office hours (see schedule on website)

2. Email professor (Include CS 277 in subject heading)

3. Talk before or after class

4. Ask questions online through Piazza or Discord

Private or not (coding)

Conceptual peers

Course Discord Link on Prairielearn

Current link invite valid for 7 days 6 days?

Strongly encouraged to join before link invalidates

Lecture channel on Discord

Lab / Course Feedback

Feedback is necessary to keep course pacing appropriate for all

Online Asynchronous Options:

Discord, Piazza, Email, Feedback Forms

In-Person:

In-class questions, labs, office hours

This is especially important in the early stages of the class.

Learning Objectives

Continue reviewing Python Fundamentals

Continue building a programming pipeline

Discuss and practice defining interfaces for computational problems

Programming Toolbox: Data Type Casting

Variables in Python are **strongly typed** and **dynamically typed**

```
x = 1.1
y = "3"
z = "4x"

print(int(x))

print(float(y))

print(int(z))

a = True
b = 5

print(a - a)

print(int(a))

print(bool(b))
```

Programming Pipeline Part 2

1. Make sure you understand the problem

What is the **input** and **output** of the problem?

Can you break the problem down into parts?

Do any of the sub-problems build off each other?

Part 1

we'll

2. Solve (and test) each part one at a time

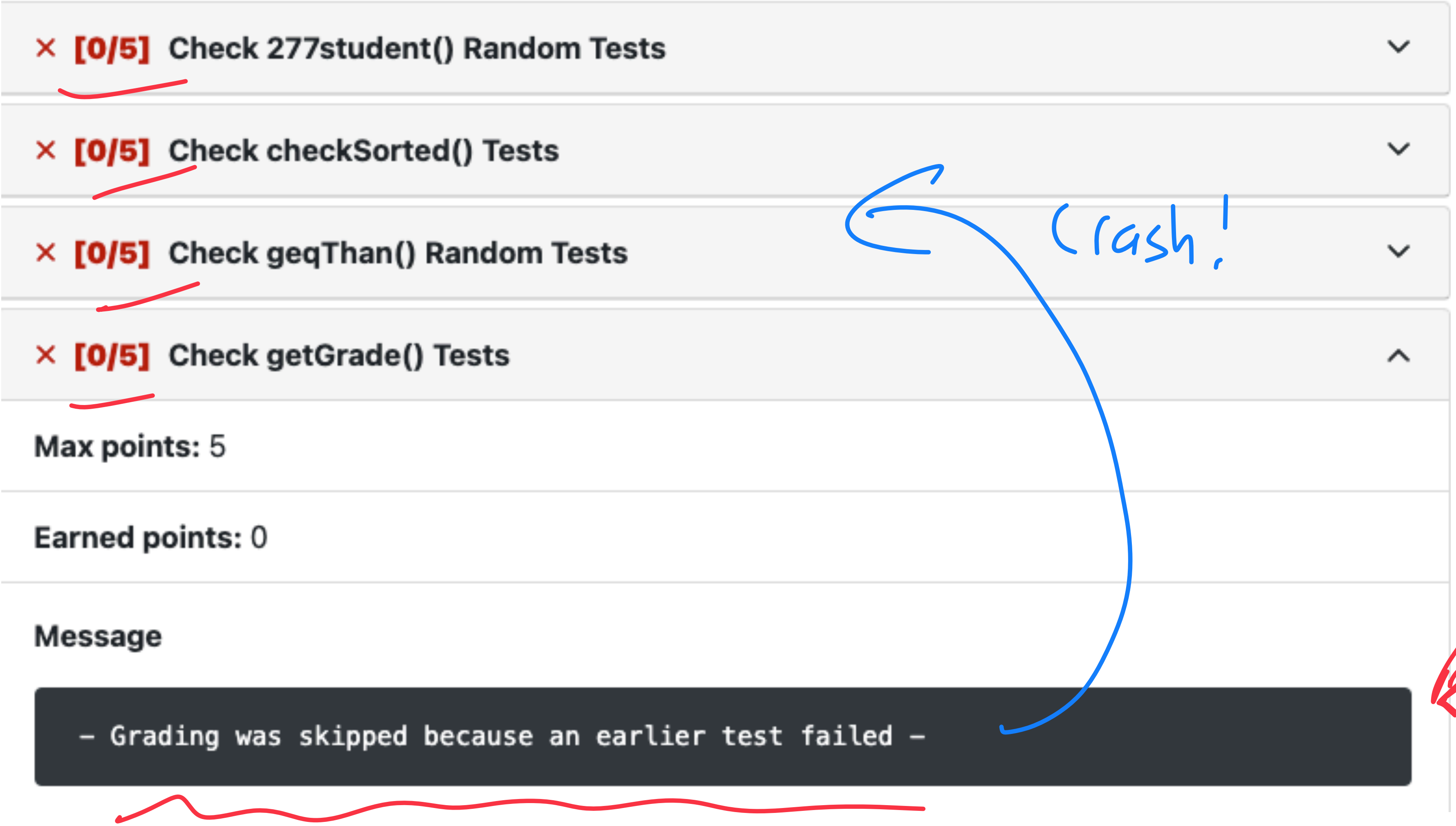
How to debug?

What should the output be given an input?

Are there any edge cases you are missing?

Debugging your Code (PrairieLearn)

"I submitted my code and didnt get points. Now what?"



× [0/5] Check 277student() Random Tests

× [0/5] Check checkSorted() Tests

× [0/5] Check geqThan() Random Tests

× [0/5] Check getGrade() Tests

Max points: 5

Earned points: 0

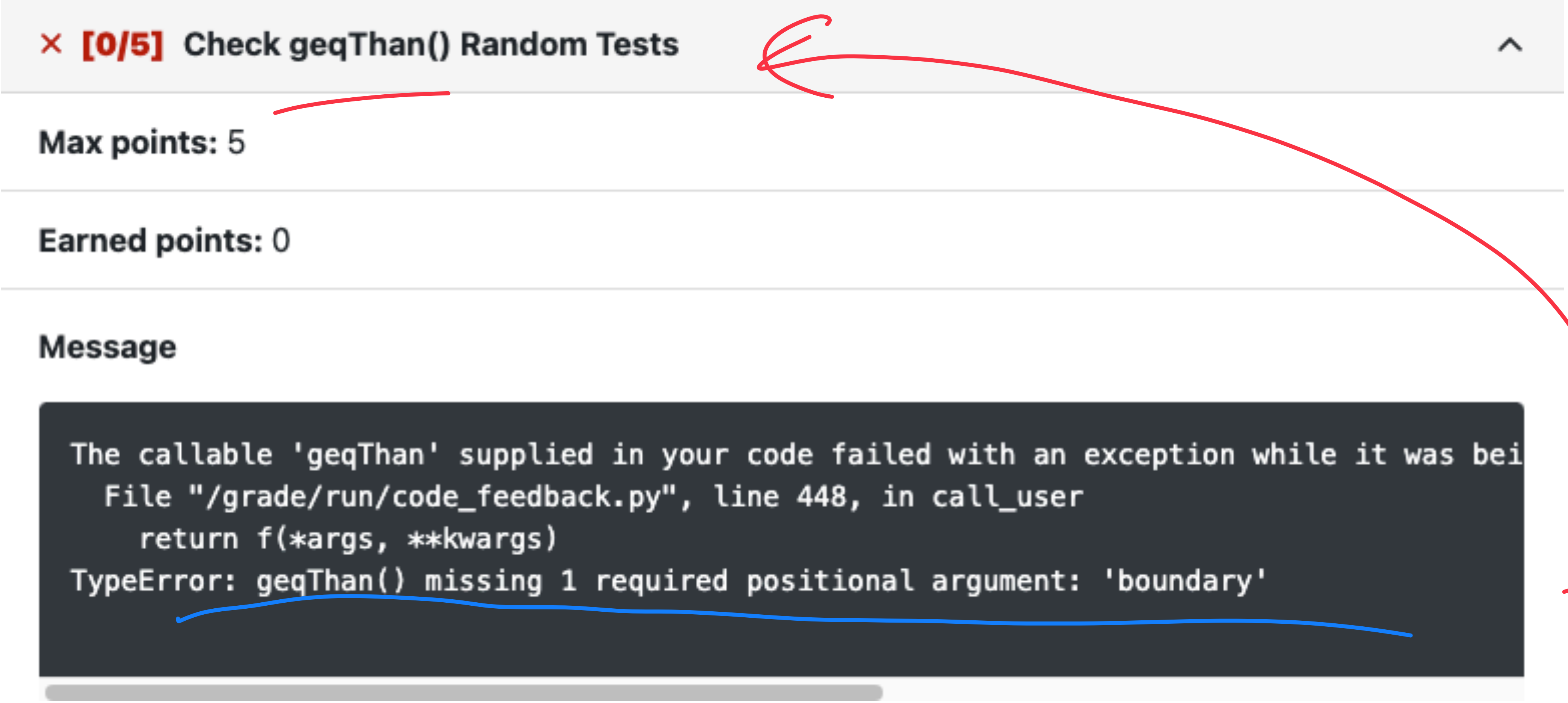
Message

```
– Grading was skipped because an earlier test failed –
```

Crash!

Debugging your Code (PrairieLearn)

Autograder is designed to give feedback on what went wrong!



× **[0/5] Check geqThan() Random Tests**

Max points: 5

Earned points: 0

Message

```
The callable 'geqThan' supplied in your code failed with an exception while it was bei
  File "/grade/run/code_feedback.py", line 448, in call_user
    return f(*args, **kwargs)
TypeError: geqThan() missing 1 required positional argument: 'boundary'
```

Debugging your Code (PrairieLearn)

getSmallestEven() has the following return message:

'Test: 55, 84, 27' is None or not defined

↳ No return

Functions / Input
— Output
↳ Default is None

electricBill() has the following return message:

'Test: 505' looks good

'Test: 49' looks good

'Test: 477' is inaccurate

↳ why did 477 fail?

1) what should eB(477) output?

2) What is your code outputting?

Python Toolbox: Print Statements

print(<string>)

↳ String formatting

x=5

```
print("ABC"+"DEF"+str(x))
```

add (plus) '+'

"A"+ "B" + str(5)

↳ ABCDEF5

Easy to use for simple strings

```
print(f"Hello {x}, its nice to meet you!")
```

format

↳ variable inside the { }

```
print("{}, {}, {}".format(1, 2, 3))
```

x y z

<String>.format(Values)

"1, 2, 3"

x+","+y+","+z

Python Toolbox: Print Statements

```
def buildString(inList):
    i=""

    for i in inList:
        i+=i

    return i

```

Programming Toolbox: Functions

Functions are defined by `def <name>(<parameters>):`

```
def getTotalTime(checkin, checkout):


def getSmallestEven(x, y, z):


def electricBill(watts):


print(getTotalTime("09:00:00","17:31:53"))


print(getSmallestEven(2, 1, 3))


print(electricBill(40))
```

Body of code

return x

def

Body

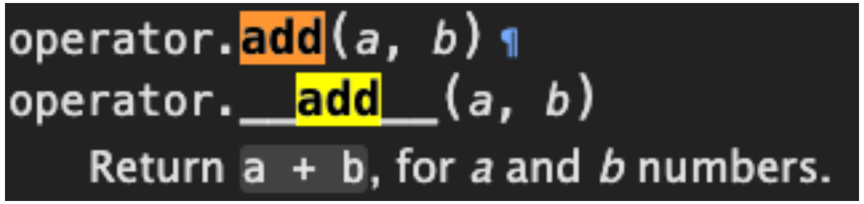
functions only run when called

Programming Toolbox: Functions

All calls are functions

Functions in Python are everywhere!

```
def f1(x, y):
    z = x + y
    return z

print(f1(1, 3))


print(f1([0, 1, 2], [3, 4, 5]))
```



```
operator.add(a, b)
operator.__add__(a, b)
    Return a + b, for a and b numbers.
```

__add__(a, b) also works for lists!

```
a = True
b = 5

print(a - a)
```

bool → int

→ 0 not float

→ 1 - 1 = 0

What did this return? Why?

Python types change

Python Toolbox: Functions

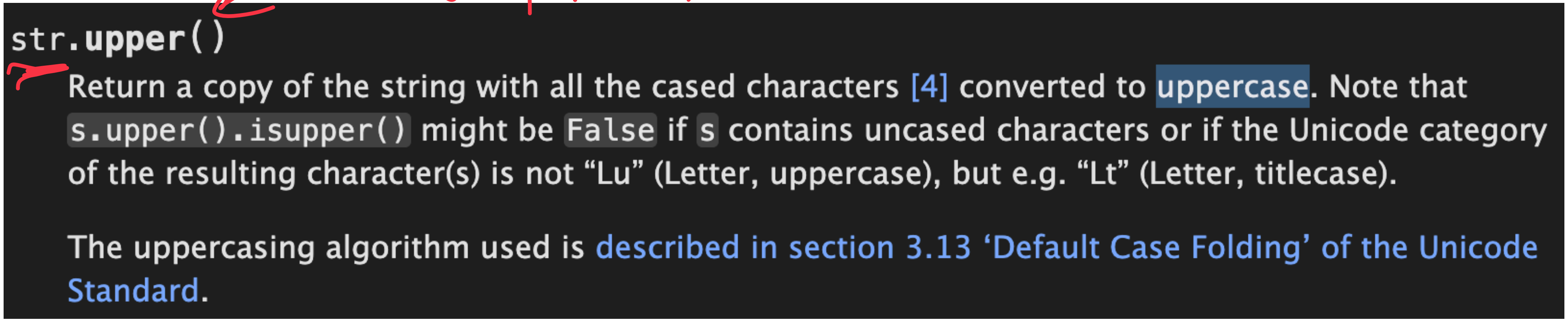
What does it mean to be the ‘**building block of programming**’?

Python is built on objects, objects are [partially] defined by functions

```
x = "string"

y = x.upper()

print(y)
print(y.lower())


```

STRING

Zero parameter



str.upper()

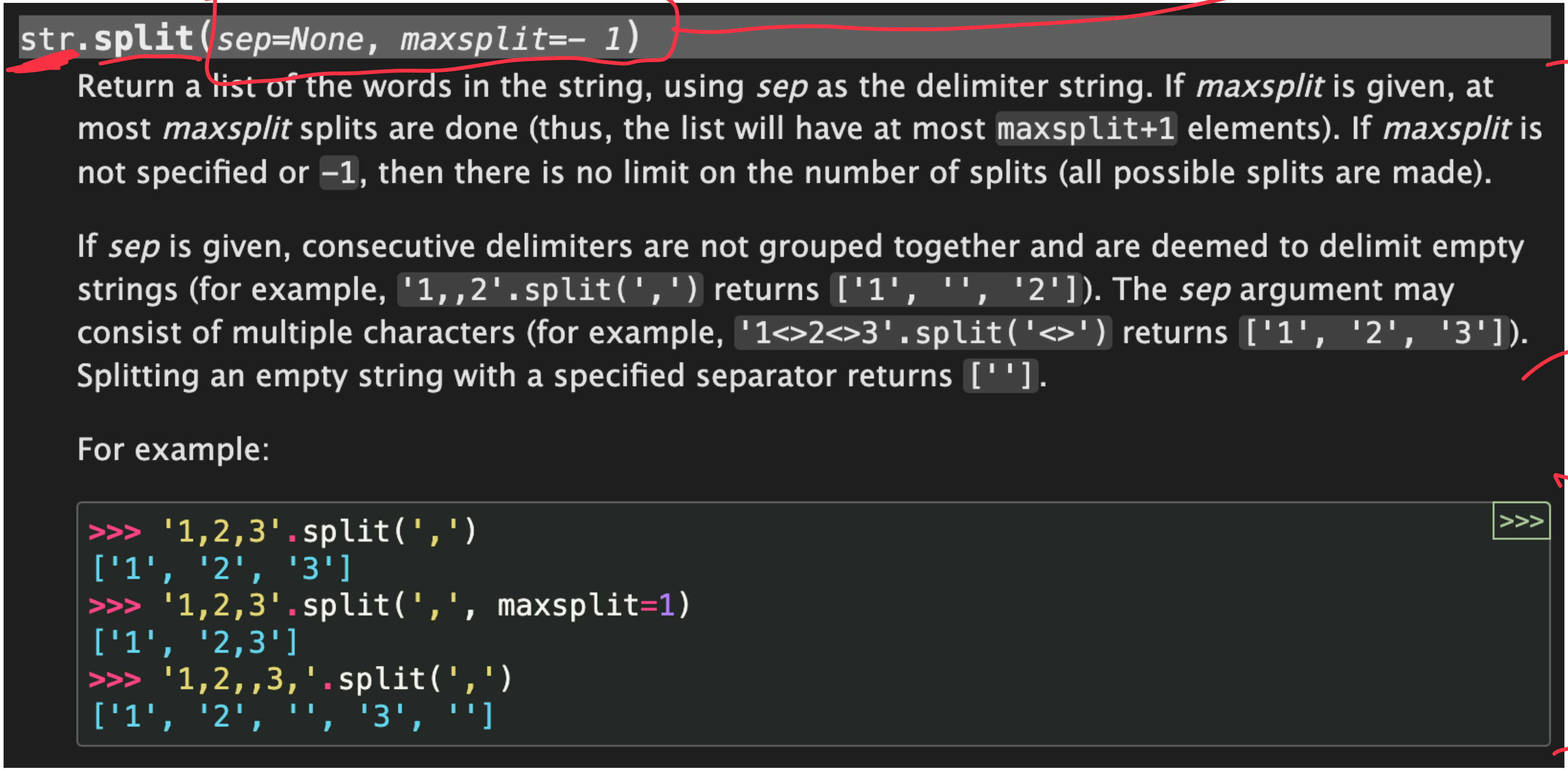
Return a copy of the string with all the cased characters [4] converted to uppercase. Note that `s.upper().isupper()` might be `False` if `s` contains uncased characters or if the Unicode category of the resulting character(s) is not “Lu” (Letter, uppercase), but e.g. “Lt” (Letter, titlecase).

The uppercasing algorithm used is described in section 3.13 ‘Default Case Folding’ of the Unicode Standard.

Python Toolbox: Functions

Default args

Learning how to read a function description is essential!



`str.split(sep=None, maxsplit=- 1)`

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done (thus, the list will have at most `maxsplit+1` elements). If *maxsplit* is not specified or `-1`, then there is no limit on the number of splits (all possible splits are made).

If *sep* is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, `'1,,2'.split(',')` returns `['1', '', '2']`). The *sep* argument may consist of multiple characters (for example, `'1<>2<>3'.split('<>')` returns `['1', '2', '3']`). Splitting an empty string with a specified separator returns `['']`.

Description

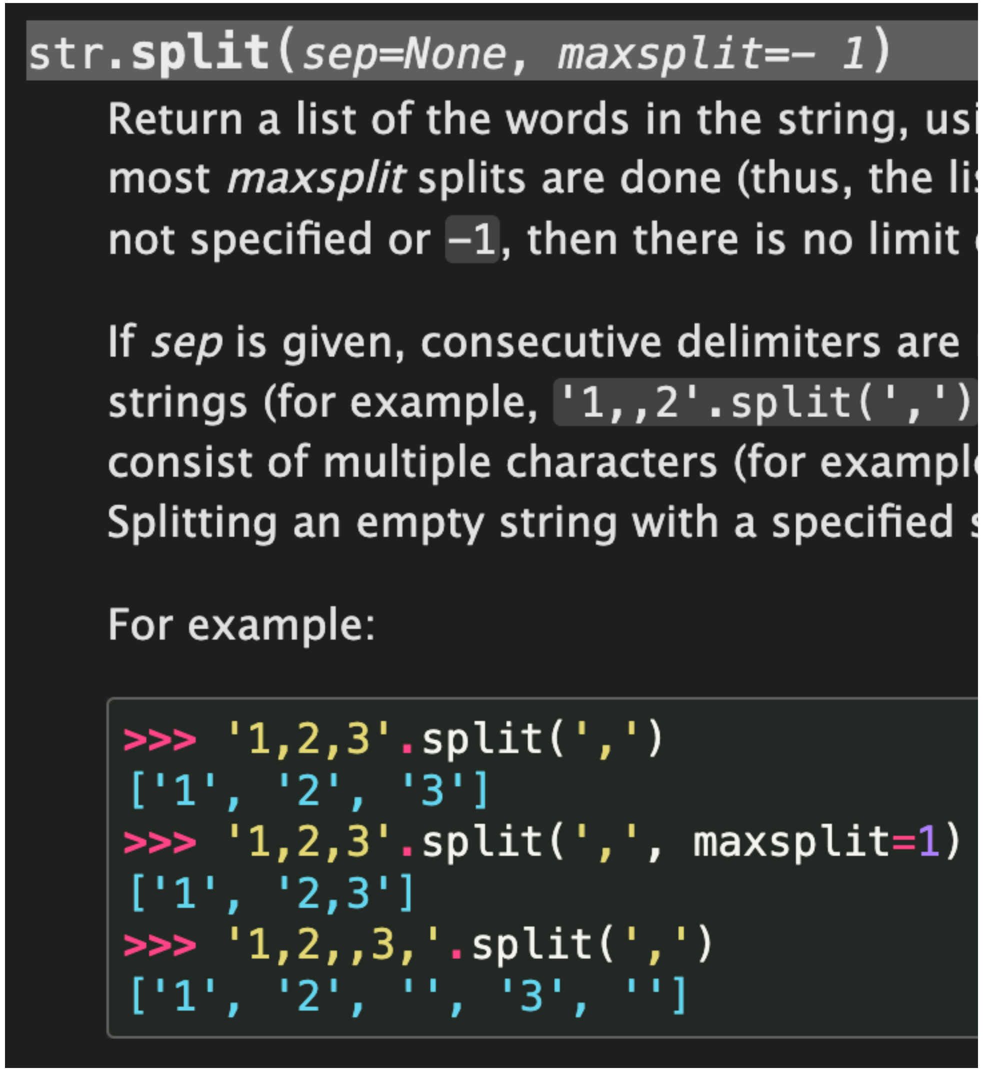
For example:

```
>>> '1,2,3'.split(',')
['1', '2', '3']
>>> '1,2,3'.split(',', maxsplit=1)
['1', '2,3']
>>> '1,2,,3,'.split(',')
['1', '2', '', '3', '']
```

Examples

Python Toolbox: Functions

Learning how to read a function description is essential!



```
str.split(sep=None, maxsplit=- 1)
```

Return a list of the words in the string, usi
most *maxsplit* splits are done (thus, the li
not specified or `-1`, then there is no limit

If *sep* is given, consecutive delimiters are
strings (for example, `'1,,2'.split(',')`
consist of multiple characters (for exampl
Splitting an empty string with a specified

For example:

```
>>> '1,2,3'.split(',')
['1', '2', '3']
>>> '1,2,3'.split(',', maxsplit=1)
['1', '2,3']
>>> '1,2,,3,'.split(',')
['1', '2', '', '3', '']
```

When in doubt — read the docs!

https://docs.python.org/3.12/

Your favorite search engine can also go a long way!

Lets practice — what does the string **strip()** function do?

↳ Removes blank spaces

whitespace from start and end

strip()

whitespace

strip("A")

lstrip()

rstrip()

Programming Practice: Functions

It is also important to be able to read a function given code → object

```
# INPUT: None
# OUTPUT: None
def f1():
    print('Function A called')

# INPUT: A Python object
# OUTPUT: The same Python object unchanged
def f2(input):
    print("Function B called")
    return input

# INPUT: A function that accepts zero args
# OUTPUT: The return value of the function
def f3(input):
    print("Function C called")
    return input()
```

None

"string"

print(f1()) → print(None)

↳ "Function A called"

↳ "None"

print(f2(5)+3)

↳ "B

print(5+3) = print(8)

↳ "8"

print(f2("Hi")+" Bye")

""

↳ "B

↳ "Hi Bye"

Programming Practice: Functions

What gets printed when running the following function call?

```
# INPUT: None
# OUTPUT: None
def f1():
    print('Function A called')

# INPUT: A Python object
# OUTPUT: The same Python object unchanged
def f2(input):
    print("Function B called")
    return input

# INPUT: A function that accepts zero args
# OUTPUT: The return value of the function
def f3(input):
    print("Function C called")
    return input()
```

None

```
print(f3(f2(f1)))
```

B called

F1 returned

C called

run f1

↳ "A

returns

None

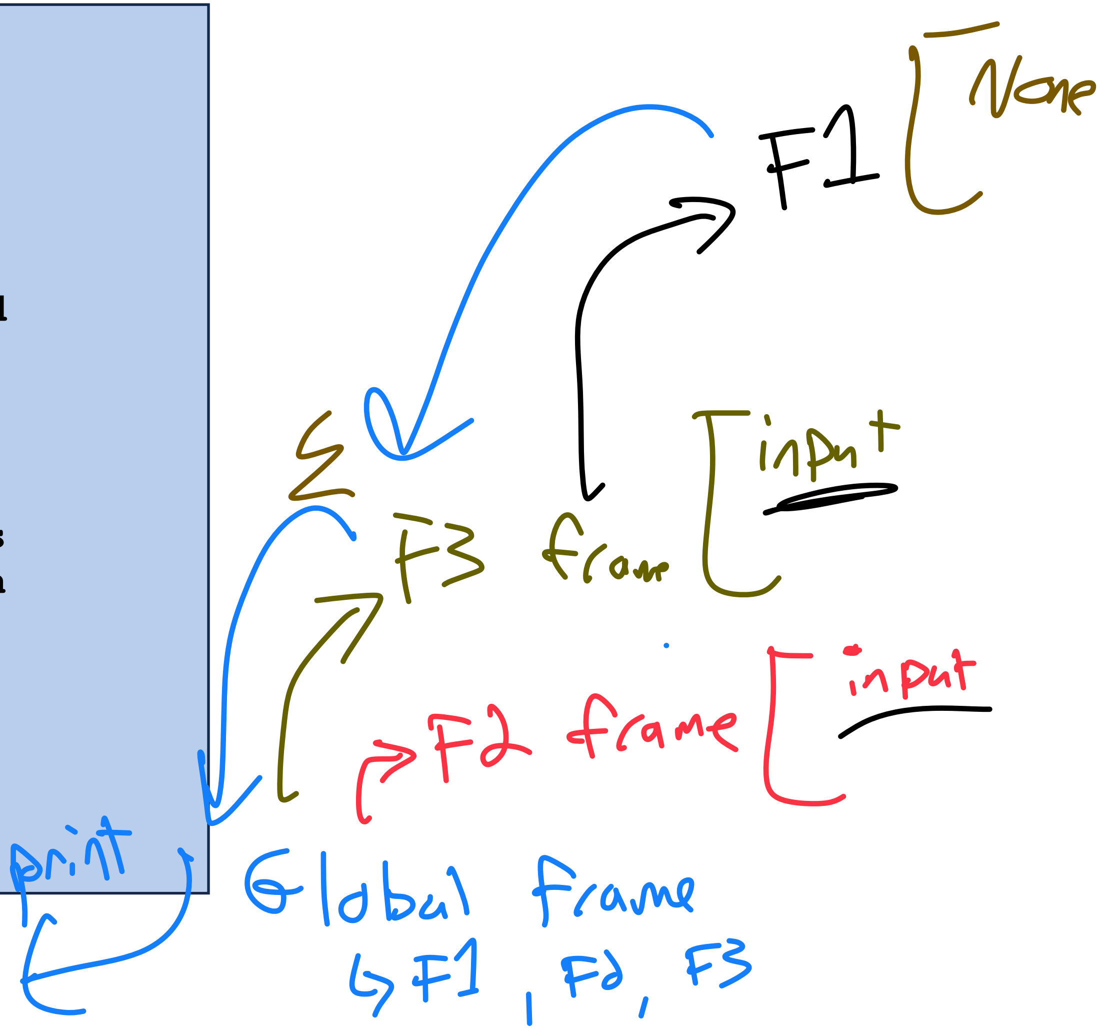
"None"

Programming Practice: Functions

Each function is its own ‘frame’ or ‘scope’

```
# INPUT: None
# OUTPUT: None
def f1():
    print('Function A called')

# INPUT: A Python object
# OUTPUT: The same Python object unchanged
def f2(input):
    print("Function B called")
    return input

# INPUT: A function that accepts zero args
# OUTPUT: The return value of the function
def f3(input):
    print("Function C called")
    return input()

print(f3(f2(f1)))
```

x= 5



Programming Practice: Functions

What happens when running the following function calls?

```
# INPUT: None
# OUTPUT: None
def f1():
    print('Function A called')

# INPUT: A Python object
# OUTPUT: The same Python object unchanged
def f2(input):
    print("Function B called")
    return input

# INPUT: A function that accepts zero args
# OUTPUT: The return value of the function
def f3(input):
    print("Function C called")
    return input()



```

```
print(f2(f1))
```

```
print(f3)
```

Programming Practice: Functions

What will the functions here print?

```
def increase(inval):
    inval+=1
    return inval

def doubleInc(inval):
    y = increase(inval)
    y += increase(inval)
    return y

print(increase(5)) # should return 6




print(doubleInc(7)) # should return 9
```

(This was secretly debugging practice)

But also scope.

see lecture recording

or

class filled in code

Programming Practice: Function Scope

```
def f1(x, y):
    x = x + y
    return x

def f2(z):
    z = [0]

def f3(z):
    z[0]=4

print(x)

a, b = 2, 5
print(f1(a, b))
print(a, b)

test = [0, 1, 2]
f2(test)
print(test)

f3(test)
print(test)

```

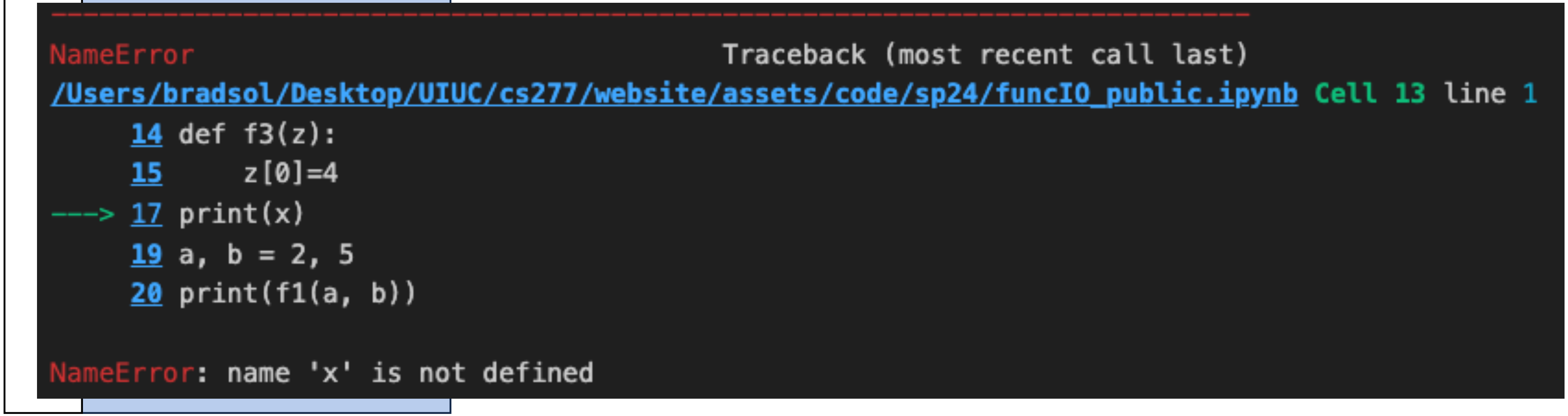
Each frame has its own variables.

Programming Practice: Function Scope

```
def f1(x, y):
    x = x + y
    return x

def f2(z):
    z = [0]

def f3(z):
    z[0]=4

print(x)

```

Each frame has its own variables.

Global **F1( )** **F2( )** **F3( )**



```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/Users/bradsol/Desktop/UIUC/cs277/website/assets/code/sp24/funcIO_public.ipynb Cell 13 line 1
     14 def f3(z):
     15     z[0]=4
---> 17 print(x)
     19 a, b = 2, 5
     20 print(f1(a, b))

NameError: name 'x' is not defined
```

Programming Practice: Function Scope

```
def f1(x, y):
    x = x + y
    return x

def f2(z):
    z = [0]

def f3(z):
    z[0]=4



a, b = 2, 5
print(f1(a, b))
print(a, b)

test = [0, 1, 2]
f2(test)
print(test)

f3(test)
print(test)

```

Each frame has its own variables.

Global	F1()	F2()	F3()
F1	x	z	z
F2	y		
F3			
a			
b			
test			

Different

Python Toolbox: Functions

Many built-in functions can take a variety of input arguments

```
import pandas

pd.read_table('myFile.csv')



pd.read_table('myFile.csv',delimiter=',')




pd.read_table(‘myFile.csv’,delimiter=',', usecols = ['Netid','Grade'])
```

Programming Toolbox: Function Overloading

Two functions are **overloaded** when they have the same name but different parameters.

```
def combine(x, y):
    return [x, y]

print(combine(5, 1))

def combine(list1, list2):
    return list1+list2

print(combine([1, 2], [3, 4]))

def combine(x, list1, list2):
    return [x]+list1+list2

print(combine(0, [1, 2], [4, 5]))
```

Programming Toolbox: Function Overloading

To properly define an overloaded function, give default arguments.

```
def combine(x, y=None, list1 = None, list2 = None):
    out = [x]
    if y:
        out+=[y]
    if list1:
        out+=list1
    if list2:
        out+=list2
    return out

print(combine(5, 1))


print(combine(0, [1, 2], [4, 5]))


print(combine(0, list1=[1, 2], list2=[4, 5]))


```

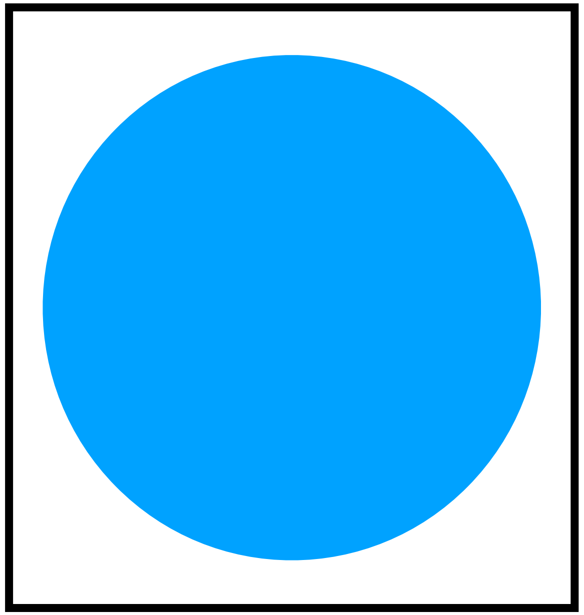
Programming Toolbox: Function Overloading

For true freedom of input, use keyword *args and **kwargs

```
def combine(*args, **kwargs):
    out = []

    for a in args:
        out.append(a)


    for k, v in kwargs.items():
        print("{} = {}".format(k, v))
        out+=v
    return out


print(combine(0, 1, 2, 3, 4, \
list1=[9, 2,3,1], list2=[8,7,2,1], \
list3 = [10]))
```

Object-Oriented Programming

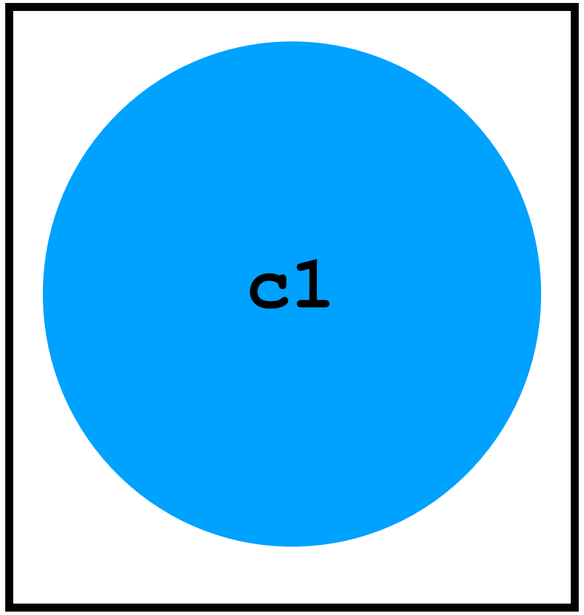
An **object** is a conceptual grouping of variables and functions that make use of those variables. A function associated with an object is a **method.**



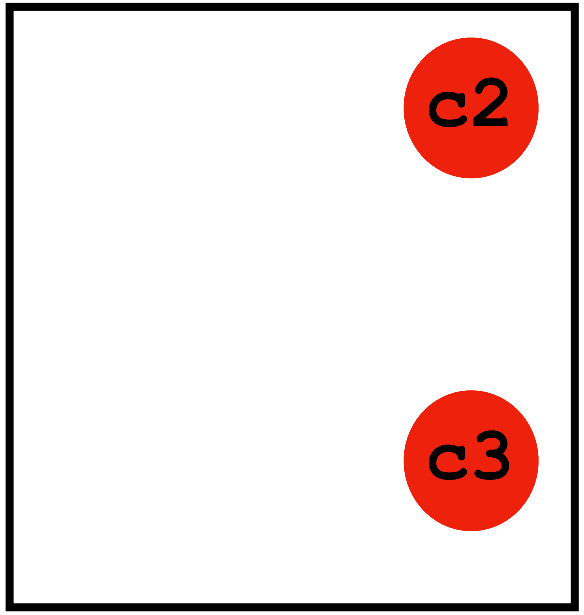
Variables:

Methods:

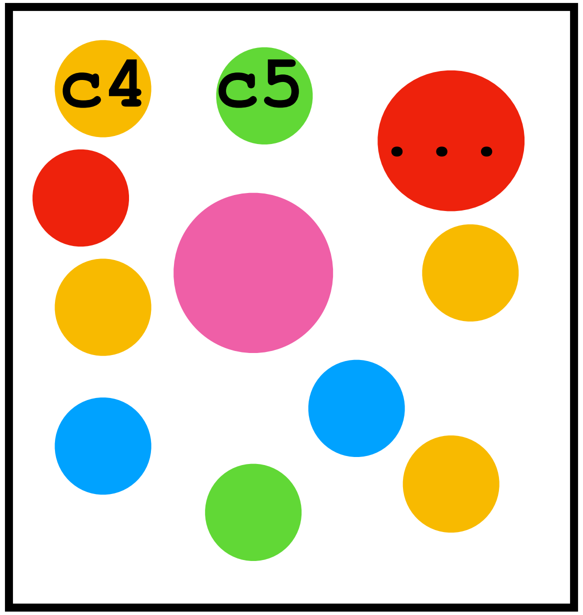
Object-Oriented Programming



```
c1.area()
```



```
c2.xpos == c3.xpos

c2.ypos == c3.ypos
```



```
getTotalArea(c4, c5, …)
```

Object-Oriented Programming

An **object** is a conceptual grouping of variables and methods that make use of those variables. ***You’ve been using these the entire time***

Everything in Python is an object

```
x = “myString"

print(x.capitalize())

print(x.find(“String"))

print(x.upper())

print(x[3]) # __getitem__()

print(x) # __str__()
```

Variables:

Type	String
Value	myString
Ref Count	1

Methods:

Object-Oriented Programming

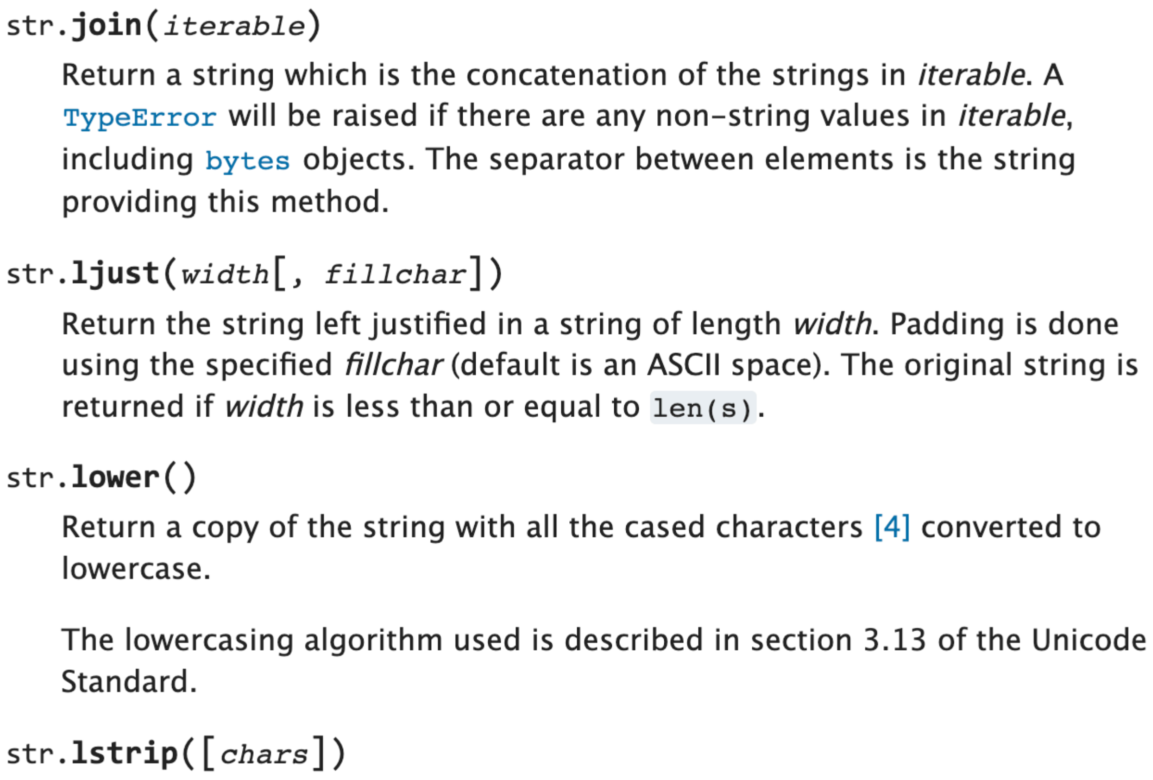
Even things that don’t have obvious function calls are (secretly) defined as a method of some object.

```
a="3"
b=3
c=3.0
d=True

print(a + b)

print(b + c)

print(c > d)
```

```
# For objects of type ‘string’
def __add__(self, o):
  ...

# For objects of type ‘int’
def __add__(self, o):
  ...

# For objects of type ‘float’
def __add__(self, o):
  ...


def __gt__(self, o):
```

Object-Oriented Programming

The collection of publicly accessible methods and variables that make up an object is its **interface.** This includes none of the implementation details.



`str.`**`join`**`(`*`iterable`*`)`

Return a string which is the concatenation of the strings in *iterable*. A `TypeError` will be raised if there are any non-string values in *iterable*, including `bytes` objects. The separator between elements is the string providing this method.

`str.`**`ljust`**`(`*`width`*`[, `*`fillchar`*`])`

Return the string left justified in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

`str.`**`lower`**`()`

Return a copy of the string with all the cased characters [4] converted to lowercase.

The lowercasing algorithm used is described in section 3.13 of the Unicode Standard.

`str.`**`lstrip`**`([`*`chars`*`])`

https://docs.python.org/3/library/stdtypes.html#string-methods

Object-Oriented Programming

We will discuss and use data structures in the context of their **interface.**

Ex: The string [data type] will have a few properties in any language

```
std::string x = "Hello World";

for(int i = x.length() - 1; i >= 0; --i){
   std::cout << x[i] << std::endl;
}
```

```
x = "Hello World"

i = len(x) - 1
while(i >= 0):
    print(x[i])
    i-=1
```

In-Class Exercise

Work with your neighbors to define an **interface** for a game of tic-tac-toe. What variables do you need? What methods would you make?