Discussion 5

www.yellkey.com/project

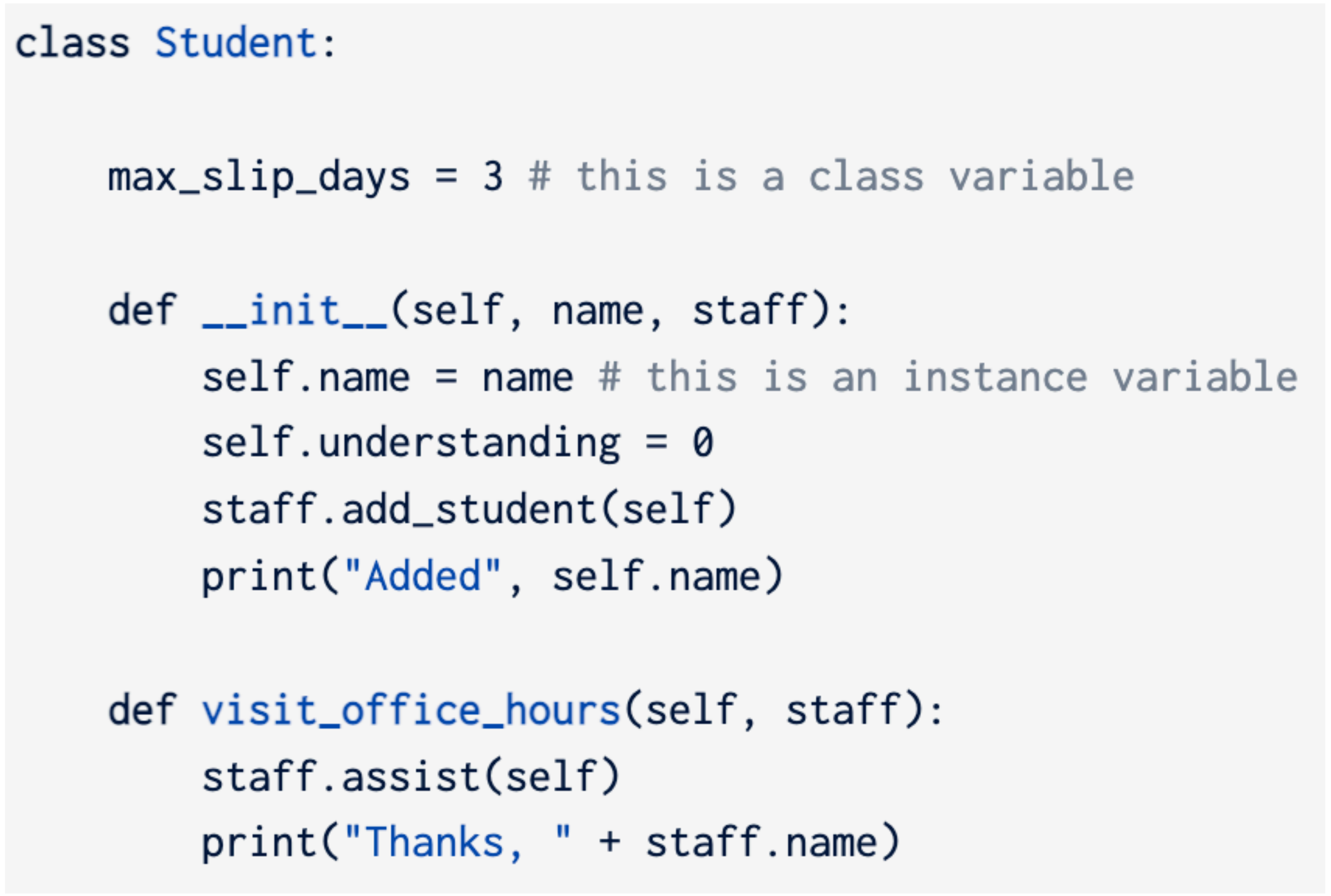
Feedback!

- Updated website

Object Oriented Programming

a programming paradigm that allows us to treat data as objects, like we do in real life.

Object Oriented Programming



```
class Student:

    max_slip_days = 3 # this is a class variable

    def __init__(self, name, staff):
        self.name = name # this is an instance variable
        self.understanding = 0
        staff.add_student(self)
        print("Added", self.name)

    def visit_office_hours(self, staff):
        staff.assist(self)
        print("Thanks, " + staff.name)
```

class
a template for creating objects

class variable
a data attribute of an object, shared by all instances of a class

Object Oriented Programming

```
class Student:

    max_slip_days = 3 # this is a class variable

    def __init__(self, name, ~~staff~~):
        self.name = name # this is an instance variable
        self.understanding = 0
        ~~staff.add_student(self)~~
        print("Added", self.name)

    def visit_office_hours(self, staff):
        ~~staff.assist(self)~~
        print("Thanks, " + staff.name)
```

instance

a single object created from a class

instance variable

a data attribute of an object, specific to an instance

How do we create an instance?

Student("Sean", ...)

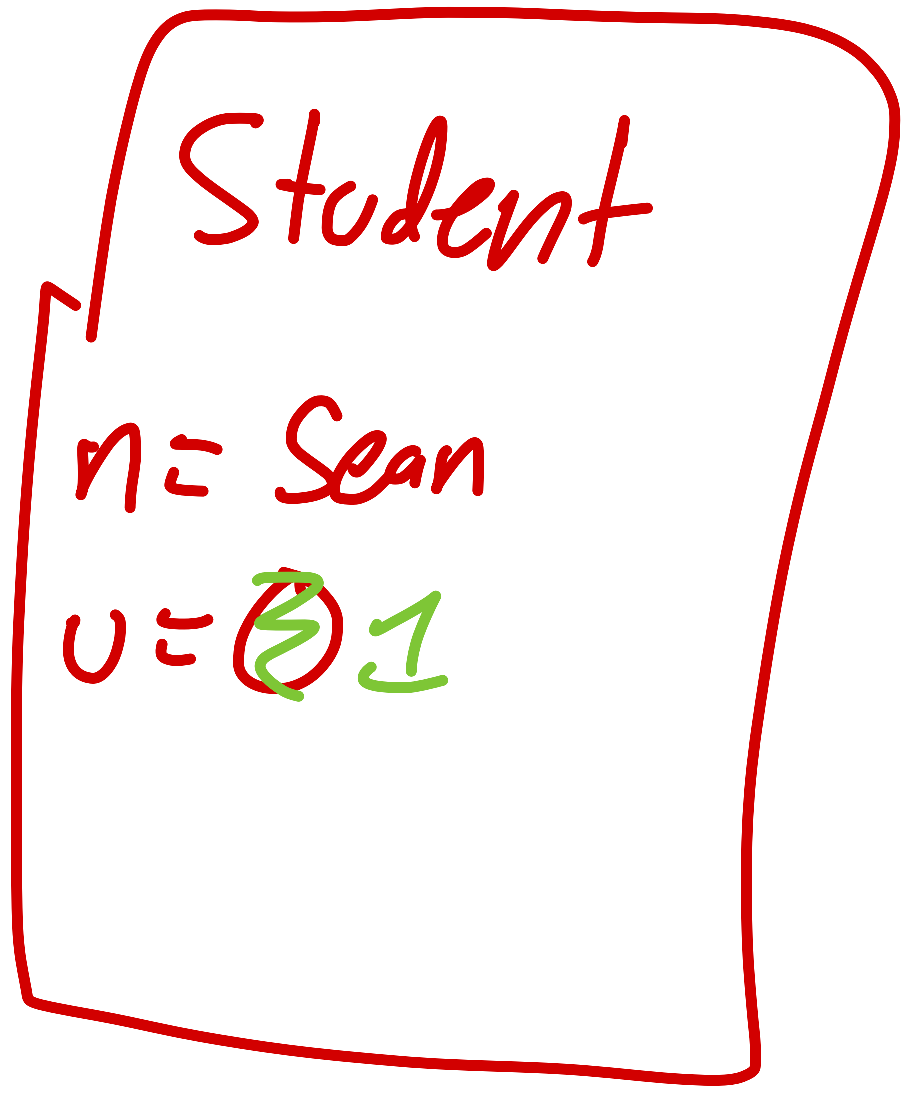
What is an init method?

Object Oriented Programming

```
class Student:

    max_slip_days = 3 # this is a class variable

    def __init__(self, name, staff):
        self.name = name # this is an instance variable
        self.understanding = 0
        staff.add_student(self)
        print("Added", self.name)

    def visit_office_hours(self, ~~staff~~):
        ~~staff.assist(self)~~      self.understanding += 1
        print("Thanks, " + staff.name)
```

method
a bound function that may be called on all instances of a class

What does visit_office_hours do?



Q3: WWPD: Student OOP

7

```
class Student:

    max_slip_days = 3 # this is a class variable

    def __init__(self, name, staff):
     →  self.name = name # this is an instance variable
     →  self.understanding = 0
     →  staff.add_student(self)   add_stude
        print("Added", self.name)

    def visit_office_hours(self, staff):
        staff.assist(self)
        print("Thanks, " + staff.name)
```

a n = e

```
class Professor:

    def __init__(self, name):
        self.name = name
        self.students = {}

    def add_student(self, student):
        self.students[student.name] = student

    def assist(self, student):
        student.understanding += 1

                                  elle     7
    def grant_more_slip_days(self, student, days):
        student.max_slip_days = days
```

Q3: WWPD: Student OOP

What will the following lines output?

```
>>> callahan = Professor("Callahan")
>>> elle = Student("Elle", callahan)
```

Added "Elle"

```
>>> elle.visit_office_hours(callahan)
```

Thanks, Callahan

```
>>> elle.visit_office_hours(Professor("Paulette"))
```

Thanks, Paulette

```
>>> elle.understanding
```

2

```
>>> [name for name in callahan.students]
```

["Elle"]

→ ["Elle"]

x = ~~lst~~ lst[3]

X = ([1,2])[0]

x = 1

Professor
name = "Callahan"
students = {
"elle":
}

Student
name = "elle"
understandings = 2
max_slip_days = 7

Professor
name = "Paulette"
students = {
}

Q3: WWPD: Student OOP

What will the following lines output?

```
>>> x = Student("Vivian", Professor("Stromwell")).name
```

Added vivian

```
>>> x
```

"vivian"

```
>>> [name for name in callahan.students]
```

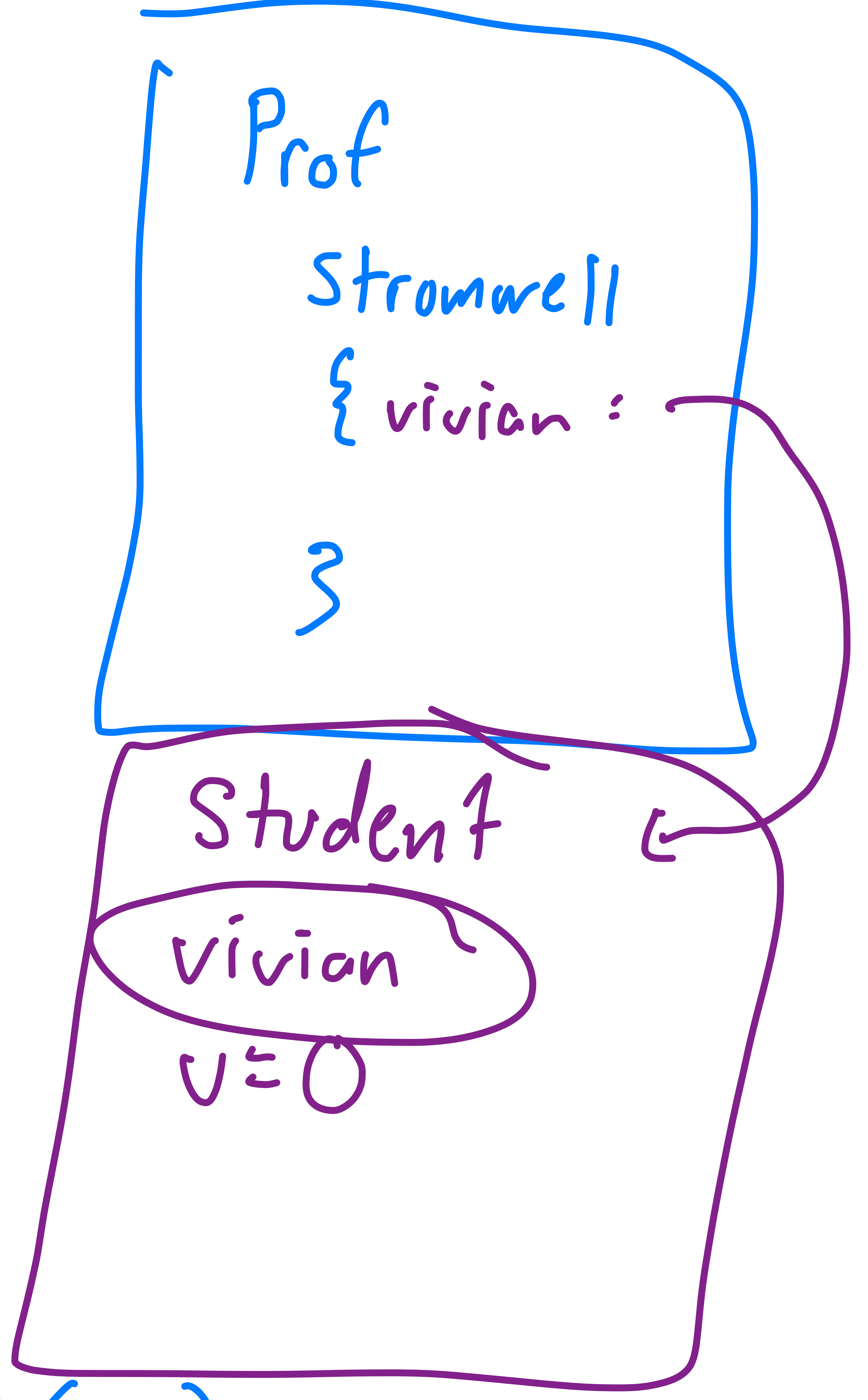
["Elle"]

```
>>> elle.max_slip_days
```

3

callahan.add_student(...)

Student.max-slip-days



Q3: WWPD: Student OOP

What will the following lines output?

```
>>> callahan.grant_more_slip_days(elle, 7)
>>> elle.max_slip_days =7
```

```
>>> Student.max_slip_days
```

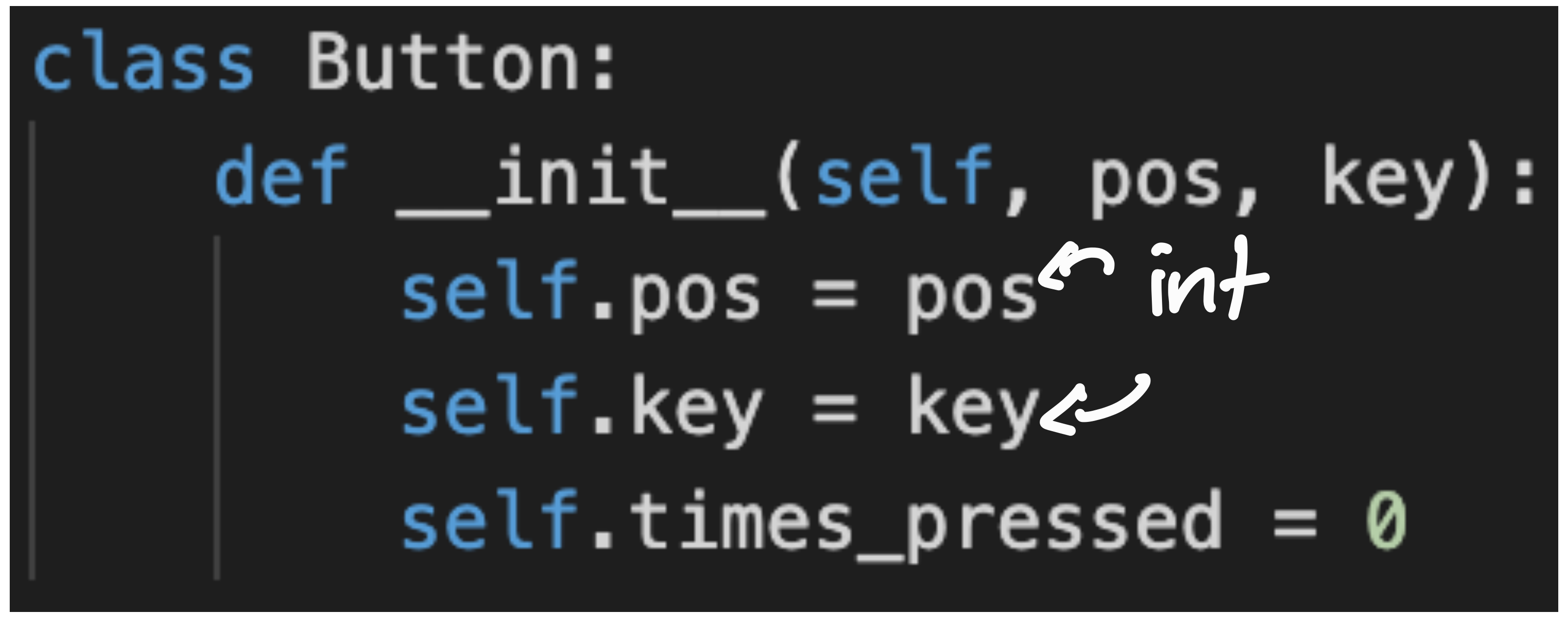
```
class Function:
    __apply(args):
        return x+2

def f(x):
    return x+2

f.explanation = "adds 2"
print(f.explanation)
```

Q4: Keyboard

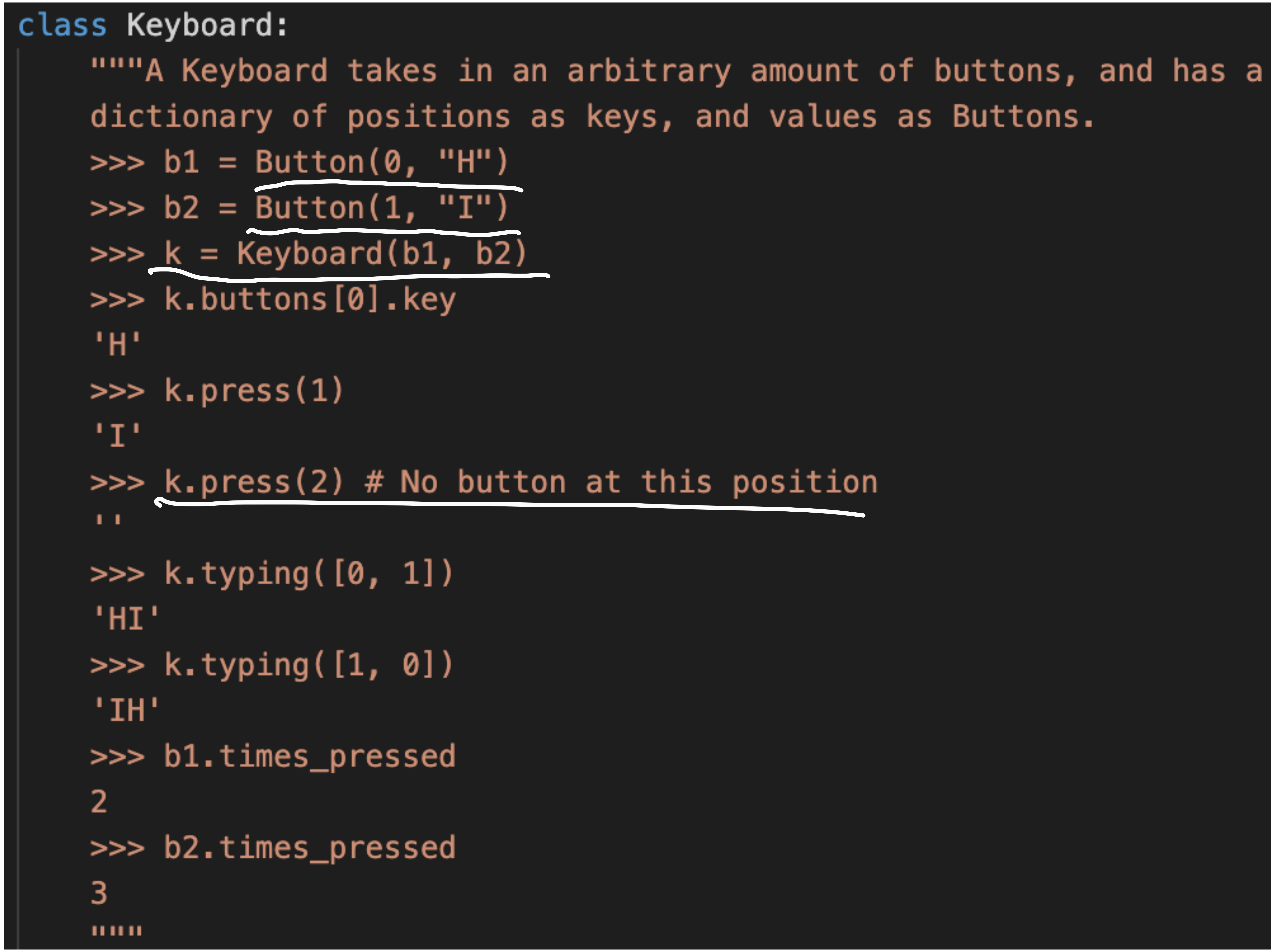
We'd like to create a `Keyboard` class that takes in an arbitrary number of `Button`s and stores these `Button`s in a dictionary.

The keys in the dictionary will be ints that represent the postition on the `Keyboard`, and the values will be the respective `Button`. Fill out the methods in the `Keyboard` class according to each description, using the doctests as a reference for the behavior of a `Keyboard`

Q4: Keyboard

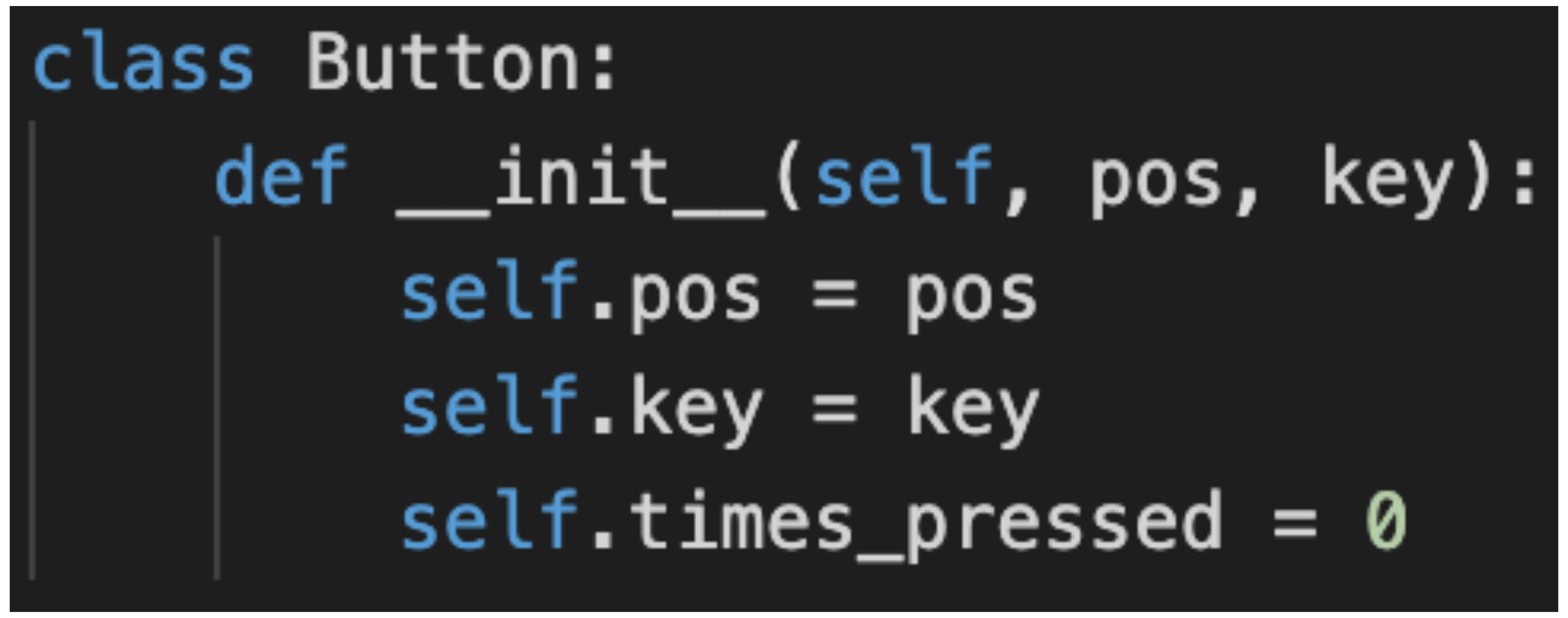


```python
class Button:
    def __init__(self, pos, key):
        self.pos = pos   int
        self.key = key
        self.times_pressed = 0
```



```python
class Keyboard:
    """A Keyboard takes in an arbitrary amount of buttons, and has a
    dictionary of positions as keys, and values as Buttons.
    >>> b1 = Button(0, "H")
    >>> b2 = Button(1, "I")
    >>> k = Keyboard(b1, b2)
    >>> k.buttons[0].key
    'H'
    >>> k.press(1)
    'I'
    >>> k.press(2) # No button at this position
    ''
    >>> k.typing([0, 1])
    'HI'
    >>> k.typing([1, 0])
    'IH'
    >>> b1.times_pressed
    2
    >>> b2.times_pressed
    3
    """
```

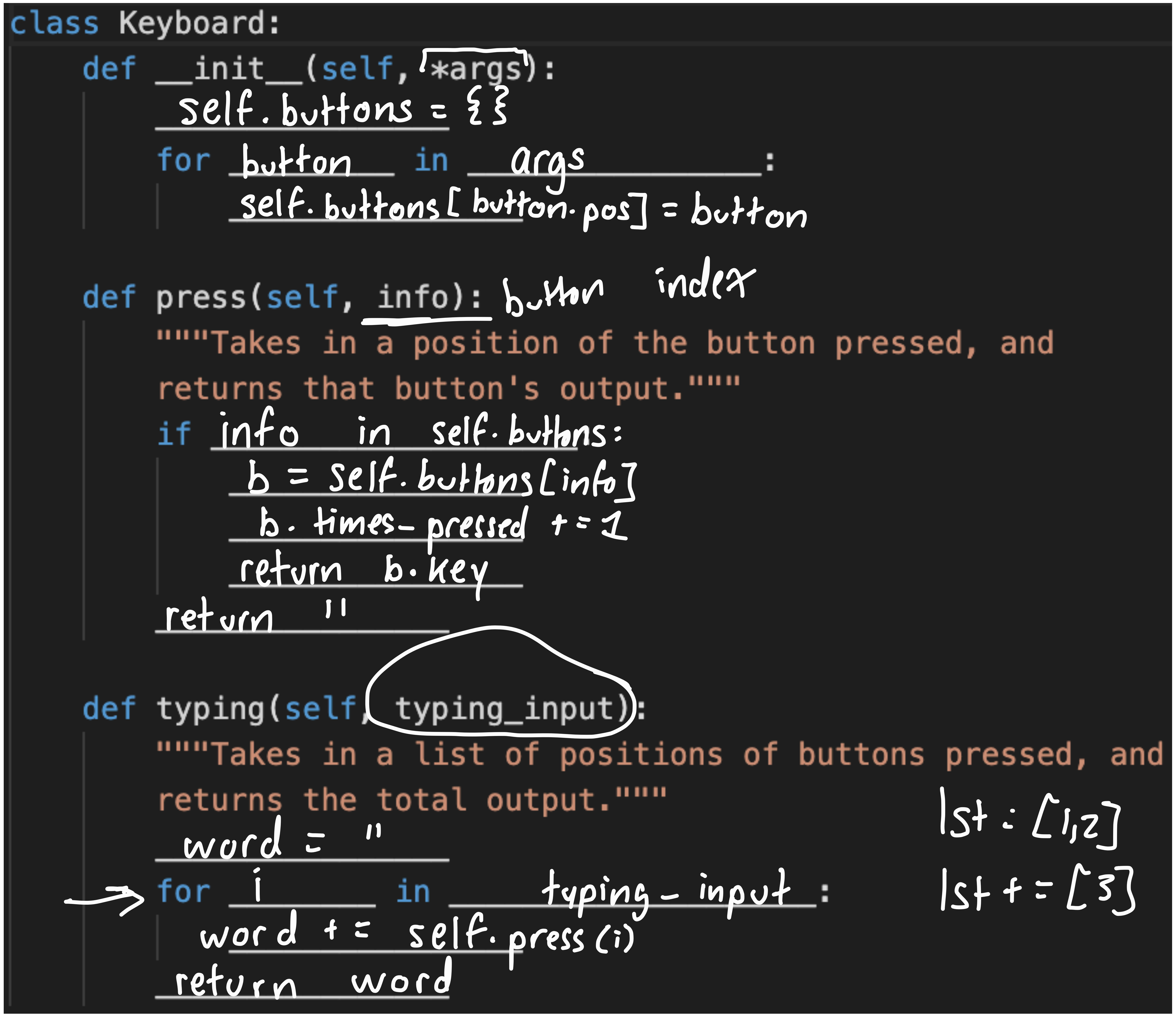
Q4: Keyboard



```
class Button:
    def __init__(self, pos, key):
        self.pos = pos
        self.key = key
        self.times_pressed = 0
```

yellkey.com/model

many objects



```
class Keyboard:
    def __init__(self, *args):
        self.buttons = {}
        for button in args:
            self.buttons[button.pos] = button

    def press(self, info): button   index
        """Takes in a position of the button pressed, and
        returns that button's output."""
        if info in self.buttons:
            b = self.buttons[info]
            b.times_pressed += 1
            return b.key
        return ''

    def typing(self, typing_input):
        """Takes in a list of positions of buttons pressed, and
        returns the total output."""                                lst : [1,2]
        word = ''
  ->    for i in typing_input:                                        lst += [3]
            word += self.press(i)
        return word
```

Q1: Map, Filter, Reduce

`my_map` takes in a one argument function `fn` and a sequence `seq` and returns a list containing `fn` applied to each element in `seq`.

```
>>> my_map(lambda x: x*x, [1, 2, 3])
[1, 4, 9]
```

Q1: Map, Filter, Reduce

`my_map` takes in a one argument function `fn` and a sequence `seq` and returns a list containing `fn` applied to each element in `seq`.

```
def my_map(fn, seq):
```

Q1: Map, Filter, Reduce

`my_filter` takes in a predicate function `pred` and a sequence `seq` and returns a list containing all elements in `seq` for which `pred` returns True.

```
>>> my_filter(lambda x: x % 2 == 0, [1, 2, 3, 4])  # new list
has only even-valued elements
[2, 4]
```

Q1: Map, Filter, Reduce

`my_filter` takes in a predicate function `pred` and a sequence `seq` and returns a list containing all elements in `seq` for which `pred` returns True.

```
def my_filter(pred, seq):
```

Q1: Map, Filter, Reduce

`my_reduce` takes in a two argument function `combiner` and a non-empty sequence `seq` and combines the elements in `seq` into one value using `combiner`.

```
>>> my_reduce(lambda x, y: x + y, [1, 2, 3, 4])  # 1 + 2 + 3 + 4
10
>>> my_reduce(lambda x, y: x * y, [1, 2, 3, 4])  # 1 * 2 * 3 * 4
24
>>> my_reduce(lambda x, y: x * y, [4])
4
```

Q1: Map, Filter, Reduce

`my_reduce` takes in a two argument function `combiner` and a non-empty sequence `seq` and combines the elements in `seq` into one value using `combiner`.

```python
def my_reduce(combiner, seq):
```

Q2: WWPD: Mutability

```
>>> s1 = [1, 2, 3]
>>> s2 = s1
>>> s1 is s2
```

```
>>> s2.extend([5, 6])
>>> s1[4]
```

```
>>> s1.append([-1, 0, 1])
>>> s2[5]
```

```
>>> s3 = s2[:]
>>> s3.insert(3, s2.pop(3))
>>> len(s1)
```

Q2: WWPD: Mutability

```
>>> s1[4] is s3[6]
```

```
>>> s3[s2[4][1]]
```

```
>>> s1[:3] is s2[:3]
```

```
>>> s1[:3] == s2[:3]
```

```
>>> s1[4].append(2)
>>> s3[6][3]
```

Feedback + Attendance

www.yellkey.com/leader

Pwd: joffreyballet