Assignment 0 remarks

1. Does `print` ever execute here? . . . . . .

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

2. What does this function return? . . . . . .

```
def Add(x ,y):
    print(x+y)
    return
```

3. What does the if/else block add to this code? . . . . . .

```
def print_dict(inp):
    if type(inp.keys()) == int:
        return list((inp.values()))
    else:
        return list((inp.keys()))
```

keys are not an integer

4. This function doesn't return what you expect, why? . . . . . .

```
def reverse_a_list(l):
    li = l.reverse()
    return li
```

list.reverse does not return anything,
thus return "l" instead
(legacy stuff from Python 2)

5. While this function works, it can be optimized . . . . . .

```
def find_digits(s):
    p = re.compile('\d')
    digits = p.findall(s)

    s = ''
    for digit in digits:
        if s == '':
            s = s + f'{digit}'
        else:
            s = s + f',{digit}'
    return s
```

fine, but use str.join instead

6. What are the coefficients stored in `model` after calling `regress?` . .

```
model = LinearRegression()
```

global function, can be overwritten

```
def regress(x,y):
    x = x.reshape(-1,1)
    reg = model.fit(x, y)
    return reg.coef_
```

local scope, overwrites the model (i.e. the coefficients will be stored within the model).

instead either:
- initiate model inside the function
- take model as an argument in the function

1. No - a function exits when it reaches a `return` statement. If the intention is to print something it should be placed before `return`. The proper way to write this function is

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

2. It returns `None`, not the expected $x + y$.

3. Nothing (except slowing down the program a bit). The `.keys` method returns a list-like object. It might be empty or hold a single item, but it is never an integer. A completely equivalent way to write this would be

```
def print_dict(inp):
    return list((inp.keys()))
```

4. The `.reverse` method does an in-place operation and returns `None`. Thus the line `li=l.reverse()` reverses `l` and sets `li` to `None`. To get the desired output we could instead do

```
def reverse_a_list(l):
    l.reverse()
    return l
```

5. Instead of manually building the string we can use `str.join`

```
def find_digits(s):
    digits = p.findall('\d', s)
    return ','.join(digits)
```

6. Calling `LinearModel()` outside the function means we use the same instance whenever we call `regress`. This alters the state of `model` even though it is defined outside of the function.