Misc Data Models

March 13, 2021

0.1 Making An Object Callable

- we can actually create an object that behaves like a function
- we can call it like a function and it does function stuff

[ ]:
```
from concurrent.futures import ProcessPoolExecutor

class Adder:
    def __init__(self, add_number):
            self.number = add_number

    def __call__(self, value):
        return self.number + value

add_one = Adder(1)
add_three = Adder(3)

numbers = [1, 2, 3, 4, 5]
with ProcessPoolExecutor() as e:
    results = e.map(add_one, numbers)
    print(list(results))
```

0.2 Clean Up With Context Managers

- make sure a process starts and stop a proceess
- basically a setup and a cleanup

```
with open('data.txt') as f:
    data = f.readline()
    print(data)
    print(f.closed)
```

[24]:
```
class Bacteria:
    def __init__(self):
        self.moving = False
        self.speed = 0
        self.direction = 90

    def move(self, amt):
```

```
        if self.moving:
            self.speed += amt

    def _turn(self, amt):
        if self.moving:
            self.direction += amt

    def turn_right(self):
        self._turn(90)

    def turn_around(self):
        self._turn(180)


    def __enter__(self):
        self.moving = True
        return self

    def __exit__(self, *args, **kwargs):
        self.moving = False
        print(f'bacteria has moved {self.speed} micrometer')
        print(f'bacteria has moved {self.direction} degrees')


with Bacteria() as bacteria:
    bacteria.move(10)
    bacteria.turn_around()
    bacteria.move(10)
    bacteria.turn_around()
```

```
bacteria has moved 20 micrometer
bacteria has moved 450 degrees
```

0.3 Copying Objects

- copy objects in memory
- sometimes you might just end up changing refrences or pointers
- a deep copy is basically just a completely new object
- `copy` module works when you a simple data structure but a more compelx data structure is alot harder

[26]:
```
from copy import copy
from copy import deepcopy

a = [1, 2, 3, 4]
b = copy(a)
a[0] = 10
```

```
c = deepcopy(a)
```

[40]:
```
from copy import copy
from copy import deepcopy

class Bacteria:
    def __init__(self, size, behavior):
        self.size = size
        self.behavior = behavior

    def __repr__(self):
        return f'<Bacteria size={self.size}>'

    def __copy__(self, *args, **kwargs):
        daughter = Bacteria(self.size, self.behavior)
        daughter.color = 'Red'
        return daughter

    def __deepcopy__(self, *args, **kwargs):
        print(args)
        print(kwargs)
        daughter = Bacteria(self.size, deepcopy(self.behavior))
        daughter.color = 'Red'

bacteria = Bacteria(5, 'mellow')

daughter = copy(bacteria)
daughter
daughter.color
```

[40]:
```
'Red'
```

0.4 Pickling Get and Set State

- if we want the information that represents an object at this moment and seralize to `binary` it so that we can store it or transmit it somewhere
- seralizing is a way of taking a complex object and putting it in a format that has a vert particular structure
- we can then `unsearlize` and put it back together
- `!!! never unpickle from the internet`

- there are times i want to pickle an object but add or remove things from it before storing it
- dumpying and loading with pickle returns two different things
- you can save data in a file to pickle it for yourself

[60]:
```
import pickle
from datetime import datetime
```

```
class Bacteria:
    def __init__(self, size, color, parent):
        self.size = size
        self.color = color
        self.parent = parent

    def __repr__(self):
        return f'<Bacteria name={self.name} title={self.title} parent={self.
 ↪parent}>'


    def __getstate__(self, *args, **kwargs):
        state = self.__dict__
        print(state)
        #del state['parent']
        state['save_data'] = datetime.now()
        print(state)
        return state


    def __setstate__(self, state):
        self.parent = []
        for piece in state:
            print(f'setting the {piece} piece to {state[piece]}')
            setattr(self, piece, state[piece])


bacteria = Bacteria(5, 'Blue', ['..', '..', '..'])


pickle.dumps(bacteria)

d = pickle.dumps(bacteria)
d
```

```
{'size': 5, 'color': 'Blue', 'parent': ['..', '..', '..']}
{'size': 5, 'color': 'Blue', 'parent': ['..', '..', '..'], 'save_data':
datetime.datetime(2021, 3, 13, 14, 20, 46, 23112)}
{'size': 5, 'color': 'Blue', 'parent': ['..', '..', '..'], 'save_data':
datetime.datetime(2021, 3, 13, 14, 20, 46, 23112)}
{'size': 5, 'color': 'Blue', 'parent': ['..', '..', '..'], 'save_data':
datetime.datetime(2021, 3, 13, 14, 20, 46, 23112)}
```

[60]:
```
b'\x80\x04\x95\x84\x00\x00\x00\x00\x00\x00\x00\x8c\x08__main__\x94\x8c\x08Bacter
ia\x94\x93\x94)\x81\x94}\x94(\x8c\x04size\x94K\x05\x8c\x05color\x94\x8c\x04Blue\
x94\x8c\x06parent\x94]\x94(\x8c\x02..\x94h\nh\ne\x8c\tsave_data\x94\x8c\x08datet
```

ime\x94\x8c\x08datetime\x94\x93\x94C\n\x07\xe5\x03\r\x0e\x14.\x00ZH\x94\x85\x94R
\x94ub.'