```
from hw3 import *
```

Assignment 3 - Object Oriented Programming

In this assignment you are going to create some classes with properties and methods and compose them together to create a command line vending machine. Since it cannot dispense real snacks, it will create files with names of snacks. To reduce the complexity of the assignment we will also forego the concept of payment and accounting. The snacks are free!

Submission:

- Use the file named `hw3.py` which has a template of the classes and some methods already implemented.
- Put your name in a comment at the top of the file.

Due Date: Monday, May 9th

In object oriented programming it is common to draw ***box models*** of the objects you will be creating. Box models usually have three sections:

1. The name of the class
2. The attributes of the class (and optionally their type, such as `string` or `int`)
3. The methods of the class

Below are the box models for this project.

We start with a generic `Item` class, which we will extend with specific types of snacks.

```
+-------------------------+
| Item                    |
+-------------------------+
| quote (class attribute) |
| name                    |
| uuid                    |
+-------------------------+
| __init__                |
| __str__                 |
| manifest                |
+-------------------------+
```

Some notes:

- `quote` will be a class attribute
- `name` is the name of the snack
- `uuid` will be a randomly assigned unique identifier
- `manifest` will create the snack as a file on disk

Below are the subclasses of `Item` that represent categories of snacks.

- `Chips`
- `Candy`
- `Nuts`
- `Pastry`
- `Gum`

Each of these will override the `quote` attribute and the `__str__` method of `Item`.

Once we have snack items, we can build a `Dispenser` class that can load and dispense snacks.

```
+-------------------------+
| Dispenser               |
+-------------------------+
| items                   |
+-------------------------+
| __init__                |
| load                    |
| dispense                |
| empty                   |
+-------------------------+
```

Notes:

- `items` will be a list
- `load` wil load one ite at a time to the *front* of the `items` list

- `dispense` will ***pop*** the front item of the list and turn it into a file (by calling the items `manifest()` method).
- `empty` will dump everything from the `items` list

And once we have a `Dispenser` class, we can create a `VendingMachine` class that has a grid of dispensers (we will create a 3x3 grid).

```
+-------------------------+
| VendingMachine          |
+-------------------------+
| REPORT_DIR              |
| TRAY                    |
| dispensers              |
+-------------------------+
| __init__                |
| load                    |
| empty                   |
| vend                    |
| count_items             |
| report                  |
| display_options         |
| run                     |
+-------------------------+
```

Notes:

- `REPORT_DIR` and `TRAY` are class attributes. They refer to directories for storing reports and snack files.
- `dispensers` is a 2-dimensional list representing the 3x3 grid of `Dispenser` objects.
- `load` will take an inventory text file, parse it, crate snack items from it, and load them into the dispenser grid
- `empty` will dump everything from each of the dispensers
- `vend` will take a coordinate pair (i.e. '00', '21', '10', etc.) that refer to the grid position and tell the dispenser at that position to dispense an item.
- `report` will create a text file listing the remaining inventory of the system
- `'display_options` will display a grid of snakcs at the fron of each dispenser
- `run` is an infinite loop method that will present the user with the command options for the vending machine.

Part 1 - The Item Class and Subclasses

I have begun the `Item` class for you. Complete it by doing the following:

1a. Write the `__str__(self)` method so that it returns a string with the following format: 'Item'

In [2]:
```
test_item = Item('pretzels')

print(test_item)
```

```
Item_pretzels_0aea1ed88e9346afbba360d4f1c70e22
```

Note that the uuid will be different for each item created.

1b. Write the `manifest(self, folder)` method so that it creates a file for the snack with the following conditions:

- the name of the file is the same as the output of `Item.__str__()`
- the file is saved in the directory passed in as `folder` (relative to where you are running the program, not from root)
  - hint: use `os.path.join` to create an OS agnostic path (you don't have to worry about whether to use forward or back slashes)
- the contents of the file contains the class `quote` string in it
- Finally, make the `manifest` method return the path of the manifested file

In [3]:
```
import os
os.mkdir('some_dir')

path = test_item.manifest('some_dir')

path
```

Out[3]:
```
'some_dir/Item_pretzels_0aea1ed88e9346afbba360d4f1c70e22'
```

In [4]:
```
with open(path, 'r') as pretzel_file:
    contents = pretzel_file.read()
    print(contents)
```

```
Enjoy!
```

1c. Now create the 5 subclasses of `Item`: `Chips`, `Candy`, `Gum`, `Pastry`, `Nuts`. Each of these will *inherit* the `__init__(self)` method from `Item` and we will not override it. But we do need to override two things...

- Override the class attribute `quote` as follows:
  - `Chips` -> `quote` = "I'm a salty snack"
  - `Candy` -> `quote` = "I rot your teeth"
  - `Gum` -> `quote` = "I freshen your breath"
  - `Pastry` -> `quote` = "I make you fat"
  - `Nuts` -> `quote` = "I'm a nutritious source of protein"
- Override the `__str__(self)` method so that 'Item' is replaced by the name of the class

In [5]:
```
skittles = Candy('Skittles')

print(skittles)
```

```
Candy_Skittles_abdfcedb428a42db80c1ee1d2db154bb
```

Part 2 - The Dispenser Class

2a. Write the `__init__(self)` method such that it creates an empty list as instance attribute called `items`

2b. Write the `load(self, item)` method such that it takes an `Item` object (or subclass) and *inserts* it to the front of the list.

```
- *Hint:* use the `insert(position, value)` value method for lists
```

2c. Write the `dispense(self, tray)` method such that:

```
- It removes the *first* item from the list
    - *hint:* use the `pop(position)` method for lists
- It makes the item manifest itself as a file in the `tray` folder
- Also, make the `dispense` method return the path of the manifested item
```

2d. Write the `empty(self)` method such that the `items` list is reset to an empty list.

In [6]:
```
test_dispenser = Dispenser()

len(test_dispenser.items)
```

Out[6]: 0

In [7]:
```
test_dispenser.load(test_item)

len(test_dispenser.items)
```

Out[7]: 1

In [8]:
```
test_dispenser.dispense('some_dir')
```

Out[8]: 'some_dir/Item_pretzels_0aea1ed88e9346afbba360d4f1c70e22'

Part 3 - The VendingMachine class

OK, now we are ready to create the `VendingMachine` class. I have created the class attributes and the `__init__` method for you.

3a. Write the `empty(self)` method such that it empties every dispenser in its `dispensers` matrix

3b. Write the `load(self, inventory)` method such that:

- It takes a file path as `inventory` and
  - opens the file
  - parses it
  - creates an item for every line
  - loads the item to the dispenser at the corresponding position
- The lines in the inventory file will have the follwing format: `<row>,<column>,<class>,<name>`
  - example1: 0,0,Chips,Doritos
  - example2: 2,1,Gum,Spearmint
- To make things easier for testing, empty all dispensers before commencing with the loading

3c. Write the `vend(self, row, column)` method such that:

- it dispenses the front item from the dispenser at the corresponding position
- it return the path of the manifested item

3d. Write the `count_items(self)` method such that it returns the total number of items in all the dispensers

3e. Write the `display_options(self)` method such that:

- It *returns* a string with the `name` of the item at the front of each dispenser
  - Note: it does not print, just returns a string. The `run()` method will do the actual printing.
- The string displays them in the same row and column arrangement (3 items on one line, followed by 3 more on another, and 3 more on the last row)
- It spaces the columns to a fixed width of 25 characters and aligned to the left
  - *hint:* use the string.format() method and use `:<25` inside the braces

3f. Write the `run(self)` method such that:

- create an infinite loop that prints a menu to screen with the follwing options:

    1. Load Machine

    1. Dispense Item

    1. Report

    1. Quit
- use `input()` to display the menu and get the user's choice
- if the user chooses '1':
  - Request the user to input a file path of an inventory file
  - Load the inventory using the `load` method
- if the user chooses '2':
  - print the `display_options` to screen and ask the user to select a snack using the coordinate position
  - use `vend` to dispense the item in the tray
- if the user chooses '3':
  - use the `report()` method (which I wrote for you, to write the current inventory to file.
- if the user chooses '4':
  - then `break` out of the infinite loop

In [11]:
```
vm = VendingMachine()
vm.run()
```

```
MENU
----------------
1) Load Machine
2) Dispense Item
3) Report
4) Quit
----------------

Enter a number: 1


Loading will clear the inventory first
Enter the inventory file path: test-inventory.txt
0 old items cleared out
90 new items successsfully loaded


MENU
----------------
1) Load Machine
2) Dispense Item
3) Report
4) Quit
----------------

Enter a number: 2


00: Doritos              01: Twinkie              02: Sun_Chips
10: Skittles             11: Peanuts              12: Snickers
20: Big_Red              21: Spearmint            22: Junior_Mints
What item would you like? 22
Your selection has been dispensed at Tray/Candy_Junior_Mints_2abbbd888f9149ec9153174dbb469936


MENU
----------------
1) Load Machine
2) Dispense Item
3) Report
4) Quit
----------------

Enter a number: 01


MENU
----------------
1) Load Machine
2) Dispense Item
```

```
3) Report
4) Quit
----------------

Enter a number: 10


MENU
----------------
1) Load Machine
2) Dispense Item
3) Report
4) Quit
----------------

Enter a number: 3


Report has been written to file Reports/Report_2016-04-25 22:11:03.193574.txt


MENU
----------------
1) Load Machine
2) Dispense Item
3) Report
4) Quit
----------------

Enter a number: 4
```

In [13]:
```
with open('Tray/Candy_Junior_Mints_2abbbd888f9149ec9153174dbb469936', 'r') as mint_file:
    print(mint_file.read())
```

```
I rot your teeth
```

Testing Your Code

I have included a test suite with this project. You can test your code as you work on it. To start, all the tests will fail. But as you implement each task, more tests will pass.

To run the test suite:

- put it and the other testing text files in the same directory as your `hw3.py` file.
- run the `hw3-test.py` file.
- It will look like this when you complete the assignment:

```
$ python3 hw3_test.py
..........
----------------------------------------------------------------------
Ran 11 tests in 0.012s
$
```

- But for now, it will show many errors, because the classes and methods have not been implemented:

```
$ python3 hw3_test.py
EE..FEFF.EF
======================================================================
ERROR: test_empty (__main__.TestDispenser)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "hw3_test.py", line 87, in test_empty
    self.assertEqual(len(self.dispenser.items), 10)
AttributeError: 'Dispenser' object has no attribute 'items'

======================================================================
ERROR: test_load_and_dispense (__main__.TestDispenser)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "hw3_test.py", line 73, in test_load_and_dispense
    self.assertEqual(len(self.dispenser.items), num+1)
AttributeError: 'Dispenser' object has no attribute 'items'

======================================================================
ERROR: test_subclass_manifests (__main__.TestItemSubclasses)
----------------------------------------------------------------------
Traceback (most recent call last):
```

```
  File "hw3_test.py", line 51, in test_subclass_manifests
    filename = str(item)
TypeError: __str__ returned non-string (type NoneType)

======================================================================
ERROR: test_report (__main__.TestVendingMachine)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "hw3_test.py", line 137, in test_report
    report_path = self.vendmac.report()
  File "/Users/jgomez/UCLA/IS271/week4/hw3.py", line 100, in report
    count = len(dispenser.items)
AttributeError: 'Dispenser' object has no attribute 'items'

======================================================================
FAIL: test_manifest (__main__.TestItemProperties)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "hw3_test.py", line 21, in test_manifest
    self.assertTrue(os.path.exists(filename))
AssertionError: False is not true

======================================================================
FAIL: test_empty (__main__.TestVendingMachine)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "hw3_test.py", line 130, in test_empty
    self.assertEqual(self.vendmac.count_items(), 90)
AssertionError: None != 90

======================================================================
FAIL: test_load (__main__.TestVendingMachine)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "hw3_test.py", line 103, in test_load
    self.assertEqual(self.vendmac.count_items(), 90)
AssertionError: None != 90

======================================================================
FAIL: test_vend (__main__.TestVendingMachine)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "hw3_test.py", line 107, in test_vend
    self.assertEqual(self.vendmac.count_items(), 90)
AssertionError: None != 90

----------------------------------------------------------------------
Ran 11 tests in 0.003s

FAILED (failures=4, errors=4)
$
```