Classes
Hashability

Making User Classes Behave (like predefined types)

- Python allows us to sort lists
  - But only if it can compare the elements
  - To make objects of a class ***sortable:***
    - Implement the __eq__ ( ) dunder
    - Implement one of __lt__( ), __le__( ), __gt__( ) or __ge__( )
    - Or even better, implement them all

Making User Classes Behave

- Keys of dictionaries need to be ***hashable***
  - A hash of an object is a large integer
  - Two objects have the same hash with only vanishingly small probability
    - This is called a collision
- A hashable class must implement:
  - Equality: __eq__(self, other)
  - Hash: __hash__(self)

Making User Classes Behave

- Defining hashes:
  - Hashes need to be integers
  - If the integers are very large, then Python only uses the lower digits of the integer

Example: Shopping Cart

- A shopping cart contains items with a certain number of quantities
- An item consists of a description and a price

```
class Item:
    def __init__(self, description, price):
        self.descr = description
        self.price = price
    def __str__(self):
        return '{} at {}'.format(self.descr, self.price
    def __repr__(self):
        return '<Item description {} price {}>'.format
            self.descr,
            self.price)
```

Example: Shopping Cart

- Selftest: We want to order items, so we need to define equality and less-than.
  - We order by the alphabetical order of the description
  - and break ties using the price

Example: Shopping Cart

- To make items hashable, we need to also implement a __hash__(self) dunder.
- Needs to return an integer
  - We can call hash on the description
    - `hash(self.descr)`
  - and multiply it with the pennies in the price
- Python will automatically cut the resulting hash down to size if necessary
- Do it

Example: Shopping Cart

```
def __eq__(self, other):
    return self.descr==other.descr and self.price==other.price

def __lt__(self, other):
    return (self.descr < other.descr or
        self.descr == other.descr and self.price < other.price)

def __hash__(self):
   return hash(self.descr)*round(100*self.price)
```

Example: Shopping Cart

- Implementing the shopping car:
  - Have list of items with associated quantities
  - Simplest implementation uses a dictionary
  - We start out with an empty dictionary
  - And implement a method add(self, item, quantity)

Example: Shopping Cart

```
class Shopping_Cart:

    def __init__(self):
        self.items = {}

def add(self, item, quantity):
    self.items[item]=quantity
```

Example: Shopping Cart

- We also need a string dunder.
  - We sort the items in the dictionary
  - We place a description of the quantity and item into a list
  - We return the join of the list, using the newline as the glue

Example: Shopping Cart

```
def __str__(self):
    lines = ['Shopping Cart:']
    for item in sorted(self.items):
     lines.append(str(self.items[item])
                  +' of '+str(item))
    return '\n'.join(lines)
```

Example: Shopping Cart

- We can also remove items from the shopping cart
  - We lower the quantity of an item in the shopping cart accordingly
  - But we raise a ValueError if we try to remove a non-existing item or more of an item than is in the shopping cart.
  - If the quantity of an item goes to zero, we remove the item completely
    - We use del to remove items from the dictionary

Example: Shopping Cart

```
def sub(self, item, quantity):
        if item in self.items:
            if quantity < self.items[item]:
                self.items[item]-= quantity
            elif quantity == self.items[item]:
                del self.items[item]
        else:
            raise ValueError
    else:
        raise ValueError
```

Programming Task

- Implement methods
  - __len__(self): returns the total number of items
  - value(self): Calculate the value of a shopping cart