COMP-SCI 5590 - 0001   Special Topics

Loop structures, List, Tuples, Set, Dictionary and Functions

Objective

- loops: for and while statements.
- Interactive loop and sentinel loop: implementations using while statement
- Nested loop
- Reading from a file in loop
- Tuples
- Set
- dictionaries
- Use case

For Loops: A Quick Review

- Suppose we want to write a program that can compute the average of a series of numbers entered by the user.

- A series of numbers could be handled by some sort of loop. If there are *n* numbers, the loop should execute *n* times.

- We need a running sum. This will use an accumulator.

For Loops

```
n = input("How many numbers do you have? ")
sum = 0.0
for i in range(n):
    x = int(input("Enter a number >> "))
    sum = sum + x
print("\nThe average of the numbers is", sum / n)
```

- Note that sum is initialized to 0.0 so that sum/n returns a float!

For Loops: Expected Result

```
How many numbers do you have? 5
    Enter a number >> 32
    Enter a number >> 45
    Enter a number >> 34
    Enter a number >> 76
    Enter a number >> 45

The average of the numbers is 46.4
```

Indefinite Loops

- We can't use a definite loop unless we know the number of iterations ahead of time.
- We can't know how many iterations we need until all the numbers have been entered.

- The *indefinite* or *conditional* loop keeps iterating until certain conditions are met.

Indefinite Loop: Example

- Here’s an example of a while loop that counts from 0 to 10:

```
i = 0
while i <= 10:
    print(i)
    i = i + 1
```

Interactive Loops

- One good use of the indefinite loop is to write *interactive loops*.
- Interactive loops allow a user to repeat certain portions of a program on demand.

Interactive Loops: Example

```
moredata = "yes"
sum = 0.0
count = 0
while moredata[0] == 'y':
   x = int(input("Enter a number >> "))
   sum = sum + x
   count = count + 1
   moredata = input("Do u have more numbers(yes or no)? ")
   print("\nThe average of the numbers is", sum / count)
```

File Loops

- Many languages don’t have a mechanism for looping through a file like this. Rather, they use a sentinel!

- We could use readline in a loop to get the next line of the file.

- At the end of the file, readline returns an empty string, “”

File Loops: Example

```
fileName = input("What file are the numbers in? ")
    infile = open(fileName,'r')
    sum = 0.0
    count = 0
    line = infile.readline()

    while line != "":
        sum = sum + int(line)
        count = count + 1
        line = infile.readline()
    print("\nThe average of the numbers is", sum / count)
```

Nested Loops

- At the top level, we will use a file-processing loop that computes a running sum and count.

- In the next level, we need to update the sum and count in the body of the loop.

- Since each line of the file contains one or more numbers separated by commas, we can split the string into substrings, each of which represents a number.

- Then we need to loop through the substrings, convert each to a number, and add it to sum.

- We also need to update count.

Nested Loops: Example

```
fileName = input("What file are the numbers in? ")
infile = open(fileName,'r')
sum = 0.0
count = 0
line = infile.readline()
while line != "":
    for xStr in line.split(","):
        sum = sum + int(xStr)
        count = count + 1
    line = infile.readline()
print("\nThe average of the numbers is", sum / count)
```

```
1,2,3
4,5,6
```

Lists

- CHANGEABLE Sequences of Data
- Syntax:

```
    list_name = [item1, item2]
```

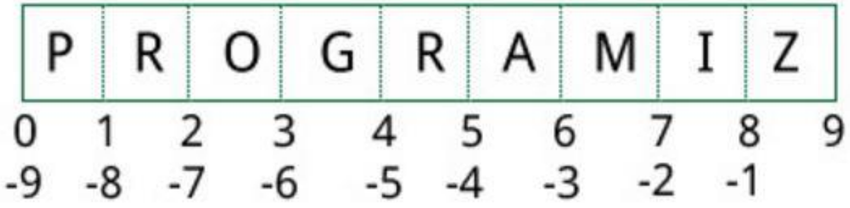
- Lists are created by using square brackets:

```
    breakfast = [ “coffee”, “tea”, “toast”, “egg” ]
```

- Indexing mechanism:

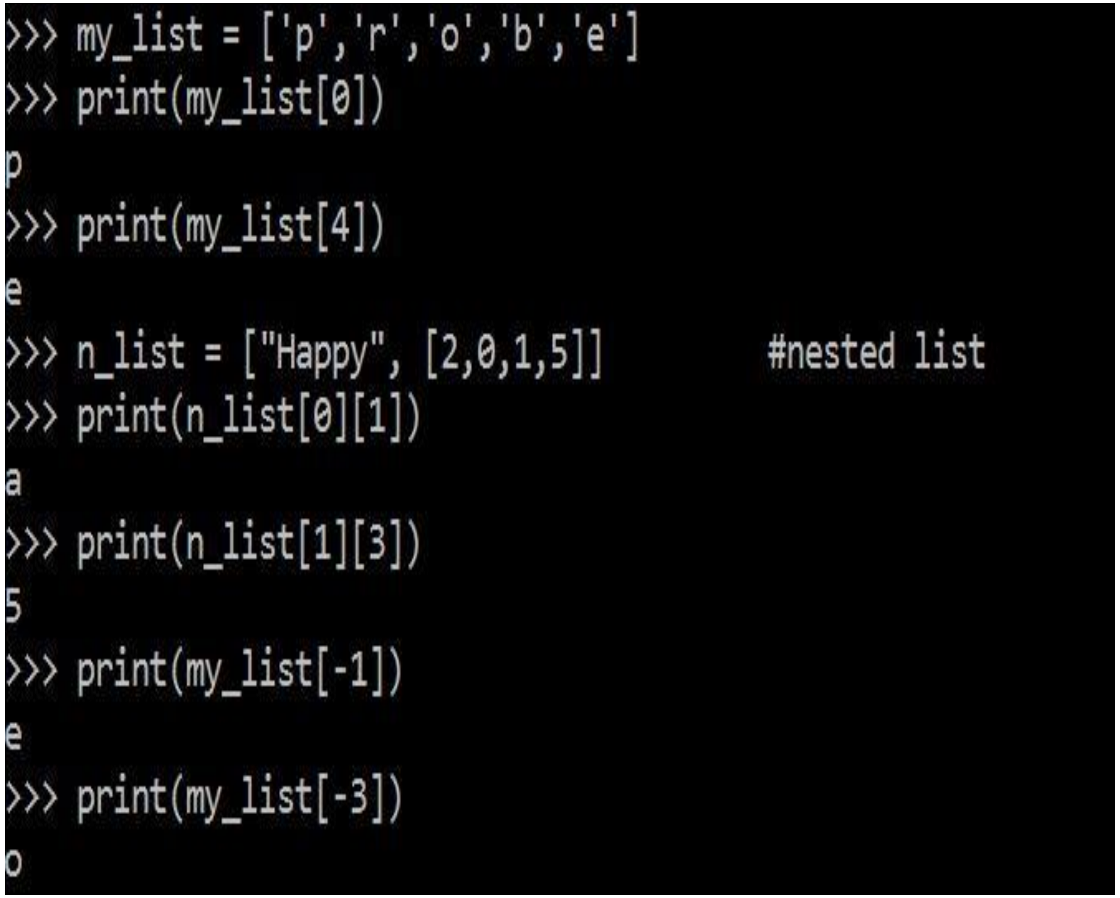
✓ It starts from 0

✓ From back it starts from -1



Indexing

- Index: a number specifying the position of an element in a list
- Enables access to individual element in list
- Index of first element in the list is 0, second element is 1, and n’th element is n-1
- Negative indexes identify positions relative to the end of the list
- The index -1 identifies the last element, -2 identifies the next to last element, etc.

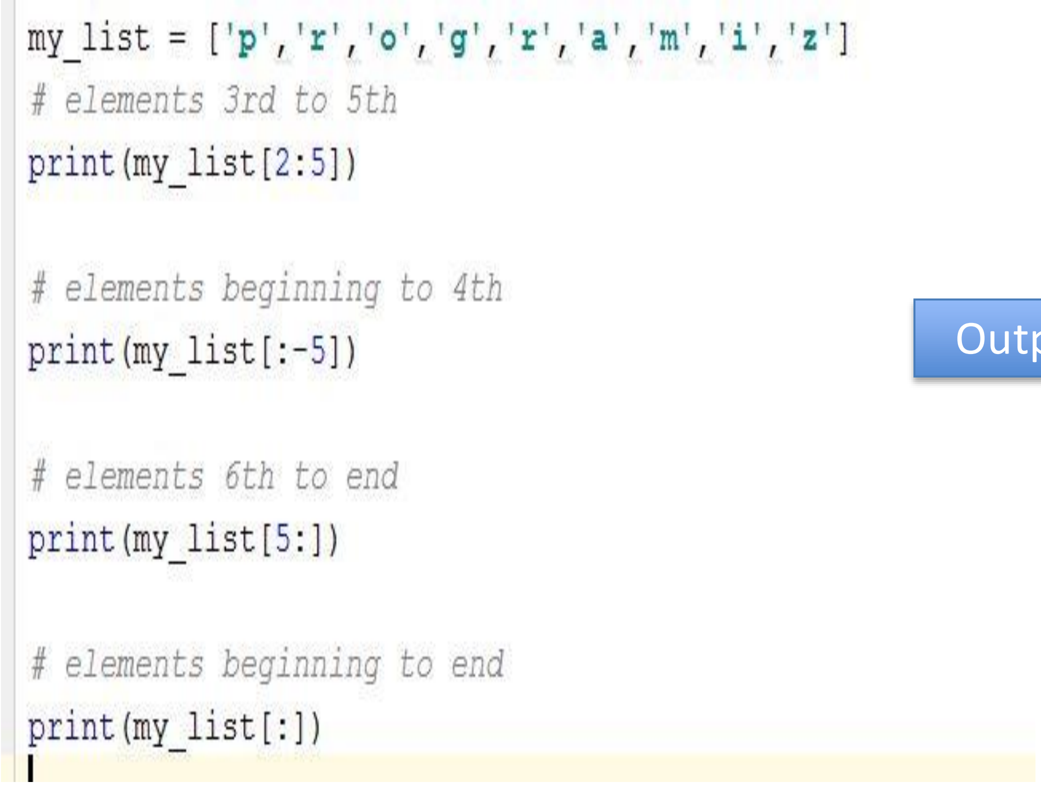


```
>>> my_list = ['p','r','o','b','e']
>>> print(my_list[0])
p
>>> print(my_list[4])
e
>>> n_list = ["Happy", [2,0,1,5]]          #nested list
>>> print(n_list[0][1])
a
>>> print(n_list[1][3])
5
>>> print(my_list[-1])
e
>>> print(my_list[-3])
o
```

List Slicing

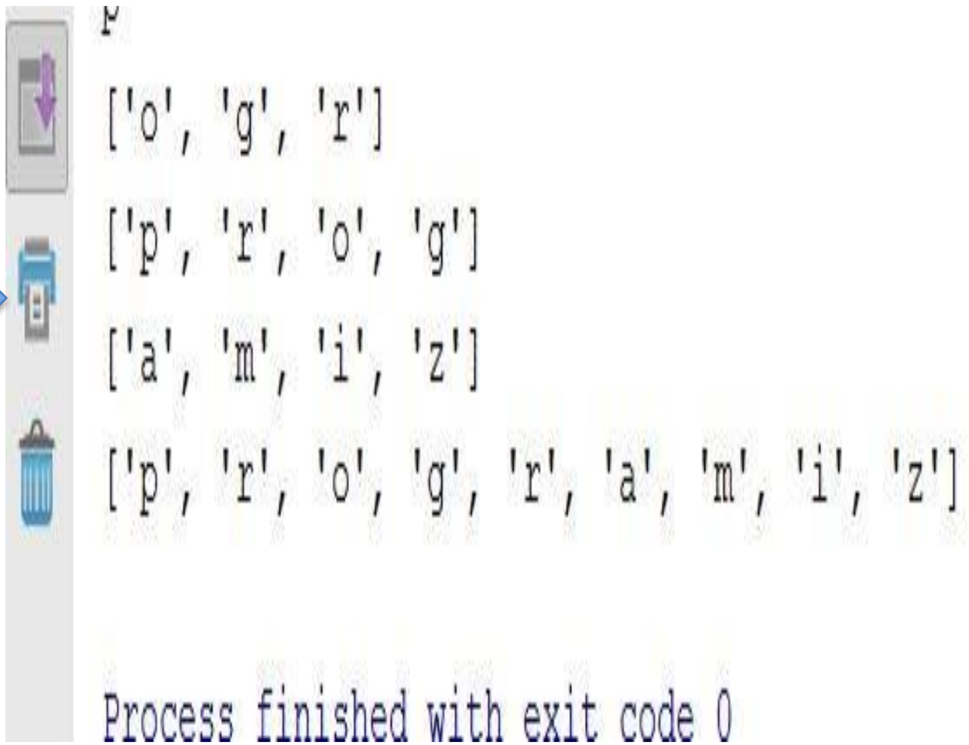
- Slice: a span of items that are taken from a sequence
  - List slicing format: *`list[start : end]`*
  - Span is a list containing copies of elements from *`start`* up to, but not including, *`end`*
    - If *`start`* not specified, `0` is used for start index
    - If *`end`* not specified, `len(list)` is used for end index
  - Slicing expressions can include a step value and negative indexes relative to end of list

Slicing Examples



```
my_list = ['p','r','o','g','r','a','m','i','z']
# elements 3rd to 5th
print(my_list[2:5])

# elements beginning to 4th
print(my_list[:-5])

# elements 6th to end
print(my_list[5:])

# elements beginning to end
print(my_list[:])
```

Output



```
p
['o', 'g', 'r']
['p', 'r', 'o', 'g']
['a', 'm', 'i', 'z']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']


Process finished with exit code 0
```

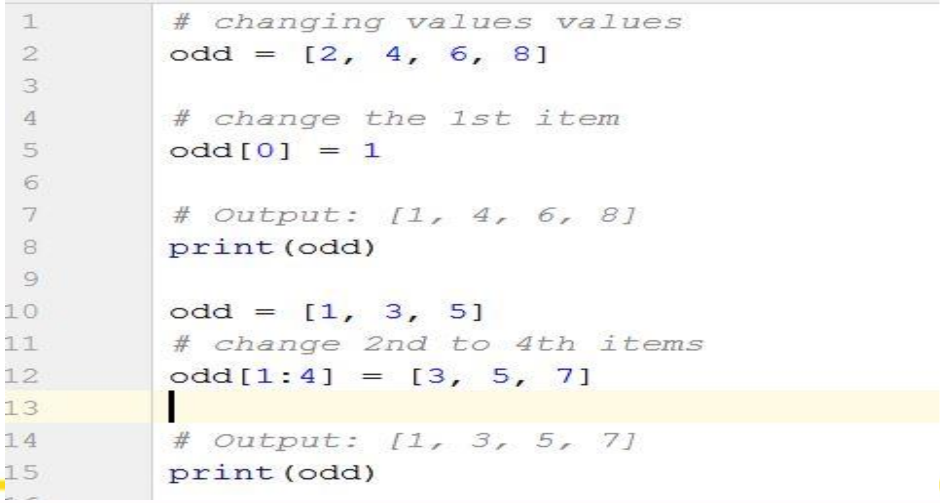
Input

List Methods and Useful Built-in Functions (cont'd.)

- insert(*index, item*): used to insert *item* at position *index* in the list
- sort(): used to sort the elements of the list in ascending order
- remove(*item*): removes the first occurrence of *item* in the list
- reverse(): reverses the order of the elements in the list
- del statement: removes an element from a specific index in a list
  - General format: del *list*[*i*]
- min and max functions: built-in functions that returns the item that has the lowest or highest value in a sequence
  - The sequence is passed as an argument

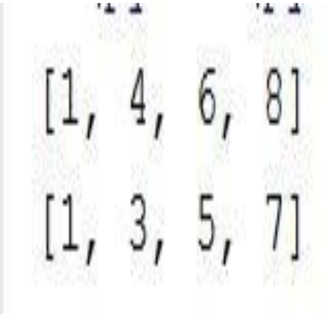
Changing Elements

- List are mutable, meaning, their elements can be changed.
- We can use assignment operator (=) to change an item or a range of items.
- `list[1] = new_value` can be used to assign a new value to a list element
- Must use a valid index to prevent raising of an `IndexError` exception



```
# changing values values
odd = [2, 4, 6, 8]

# change the 1st item
odd[0] = 1

# Output: [1, 4, 6, 8]
print(odd)

odd = [1, 3, 5]
# change 2nd to 4th items
odd[1:4] = [3, 5, 7]

# Output: [1, 3, 5, 7]
print(odd)
```

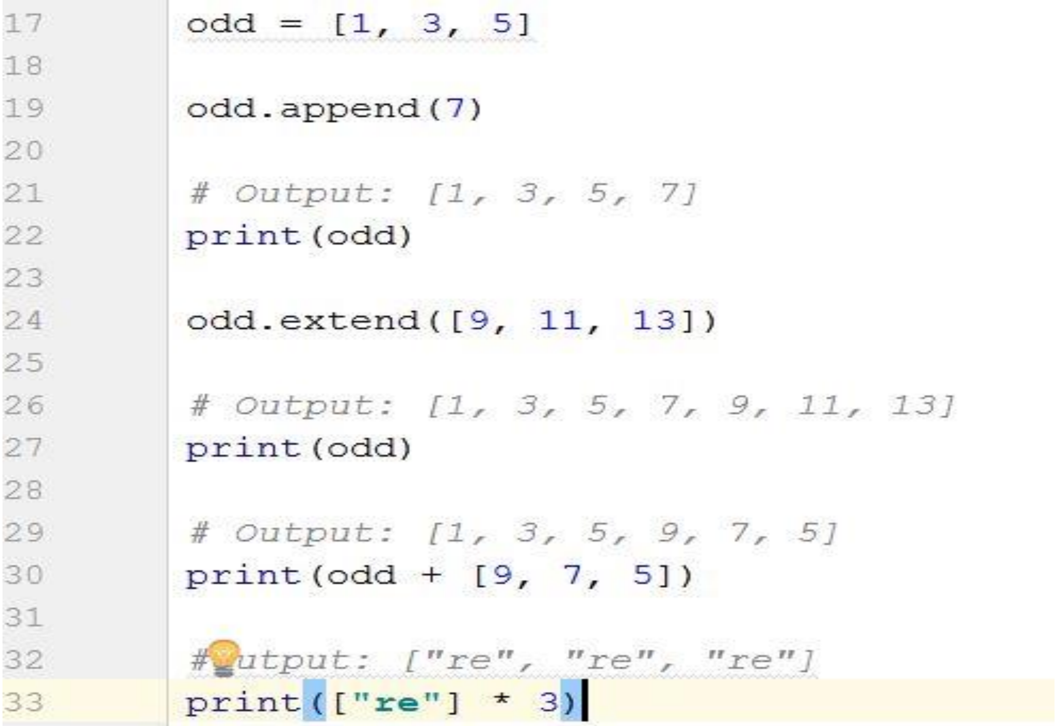
Output



```
[1, 4, 6, 8]
[1, 3, 5, 7]
```

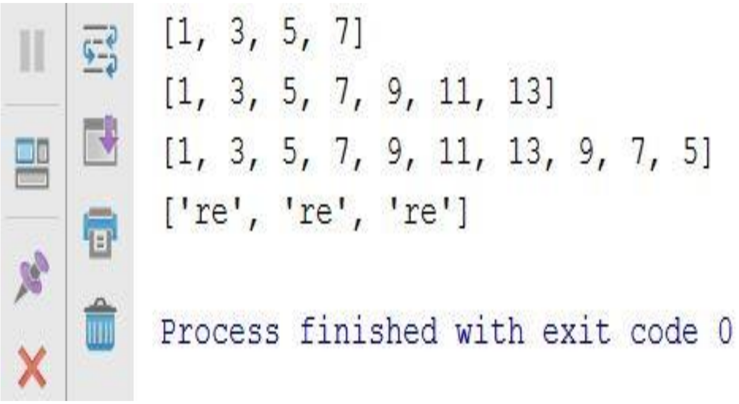
Addition

- We can add one item to a list using append() method or add several items using extend()method



```
odd = [1, 3, 5]

odd.append(7)

# Output: [1, 3, 5, 7]
print(odd)

odd.extend([9, 11, 13])

# Output: [1, 3, 5, 7, 9, 11, 13]
print(odd)

# Output: [1, 3, 5, 9, 7, 5]
print(odd + [9, 7, 5])

#utput: ["re", "re", "re"]
print(["re"] * 3)
```

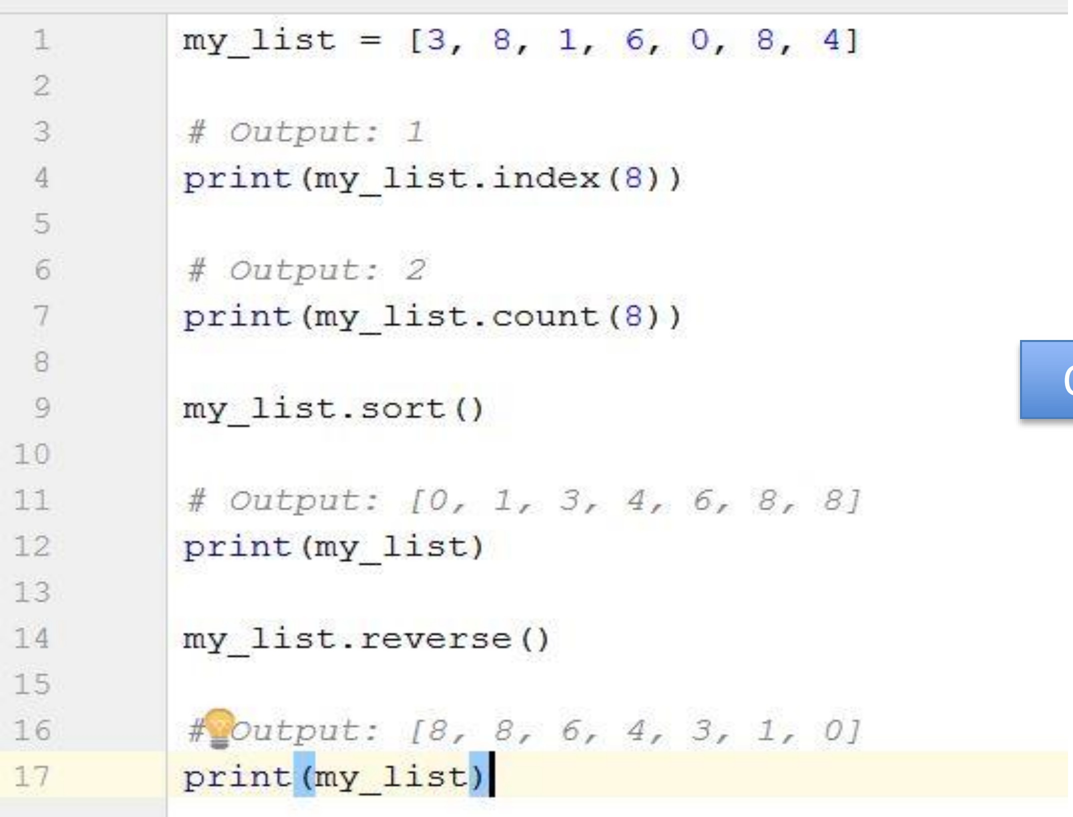
Output



```
[1, 3, 5, 7]
[1, 3, 5, 7, 9, 11, 13]
[1, 3, 5, 7, 9, 11, 13, 9, 7, 5]
['re', 're', 're']

Process finished with exit code 0
```

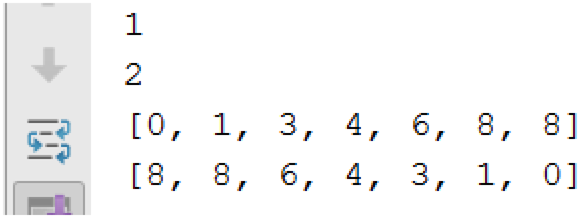
- We can also use + operator to combine two lists. This is also called concatenation.

Other built in function Examples



```
my_list = [3, 8, 1, 6, 0, 8, 4]

# Output: 1
print(my_list.index(8))

# Output: 2
print(my_list.count(8))

my_list.sort()

# Output: [0, 1, 3, 4, 6, 8, 8]
print(my_list)

my_list.reverse()

# Output: [8, 8, 6, 4, 3, 1, 0]
print(my_list)
```

Output



```
1
2
[0, 1, 3, 4, 6, 8, 8]
[8, 8, 6, 4, 3, 1, 0]
```

Deletion

```
mylist = ['p','k','h','a','o']
del mylist[0]
print(mylist)

del mylist[2:3]
print(mylist)
```

Output

```
['k', 'h', 'a', 'o']
['k', 'h', 'o']
```

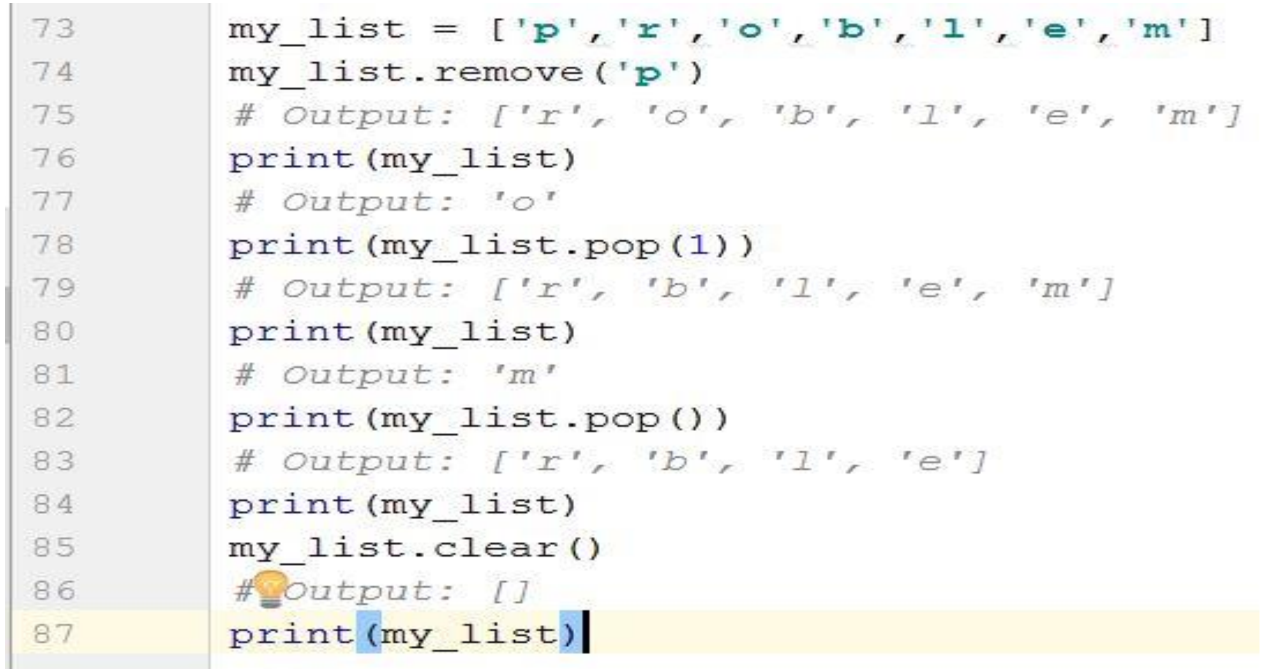
Continued…

- We can use remove() method to remove the given item or pop() method to remove an item at the given index.

- The pop() method removes and returns the last item if index is not provided.

- This helps us implement lists as stacks (first in, last out data structure).

- We can also use the clear() method to empty a list.

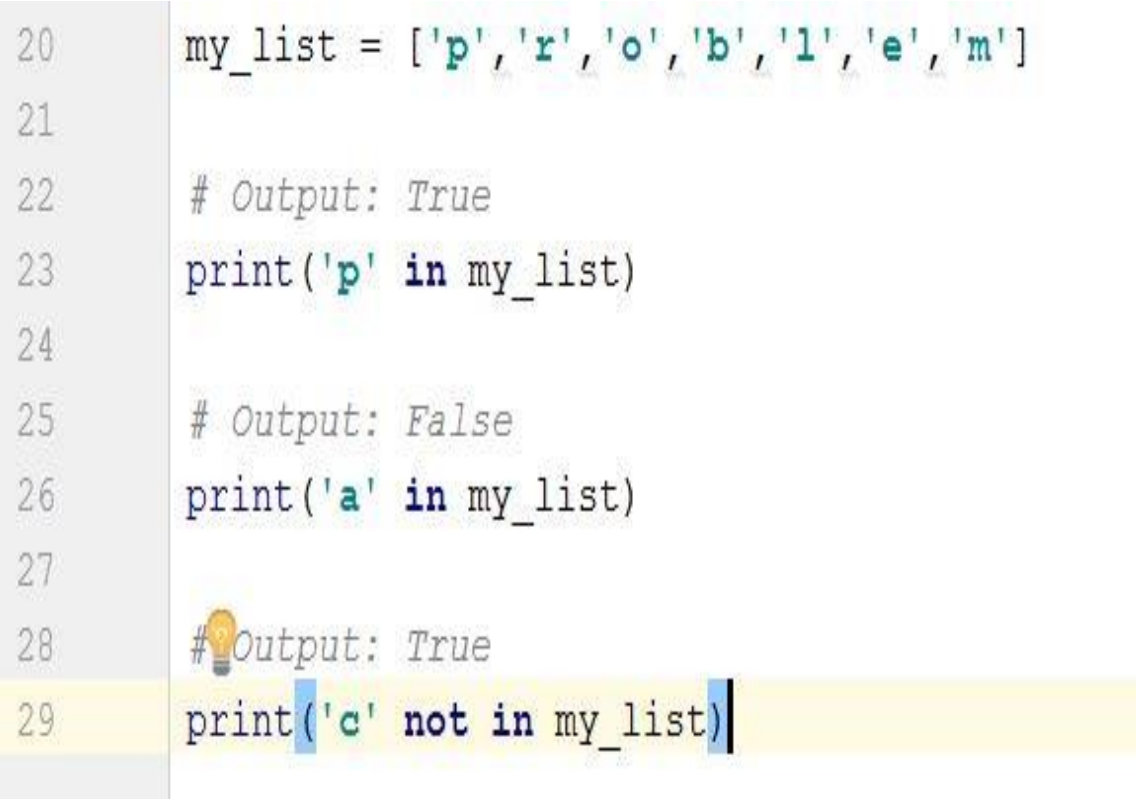


```python
my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')
# Output: ['r', 'o', 'b', 'l', 'e', 'm']
print(my_list)
# Output: 'o'
print(my_list.pop(1))
# Output: ['r', 'b', 'l', 'e', 'm']
print(my_list)
# Output: 'm'
print(my_list.pop())
# Output: ['r', 'b', 'l', 'e']
print(my_list)
my_list.clear()
#Output: []
print(my_list)
```

Finding Items in Lists with the `in` Operator

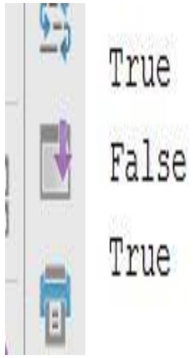
- You can use the `in` operator to determine whether an item is contained in a list
  - General format: `item in list`
  - Returns `True` if the item is in the list, or `False` if it is not in the list
- Similarly you can use the `not in` operator to determine whether an item is not in a list

Continued…



```
my_list = ['p','r','o','b','l','e','m']

# Output: True
print('p' in my_list)

# Output: False
print('a' in my_list)

# Output: True
print('c' not in my_list)
```

Output



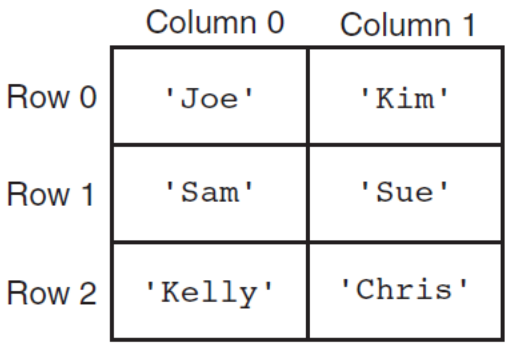
```
True
False
True
```

Two-Dimensional Lists

- Two-dimensional list: a list that contains other lists as its elements
  - Also known as nested list
  - Common to think of two-dimensional lists as having rows and columns
  - Useful for working with multiple sets of data
- To process data in a two-dimensional list need to use two indexes
- Typically use nested loops to process

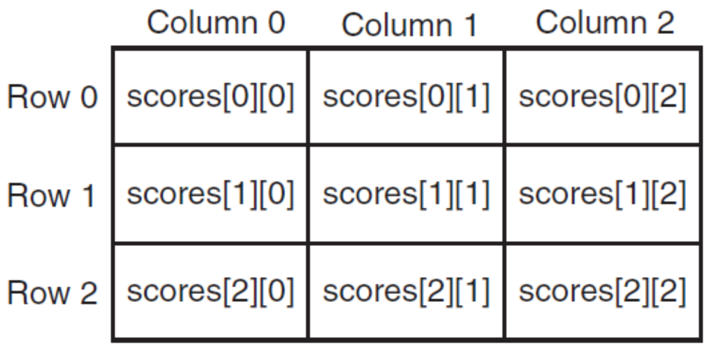
Two-Dimensional Lists (cont’d.)

Figure 7-5 A two-dimensional list



	Column 0	Column 1
Row 0	'Joe'	'Kim'
Row 1	'Sam'	'Sue'
Row 2	'Kelly'	'Chris'

Figure 7-7 Subscripts for each element of the `scores` list



	Column 0	Column 1	Column 2
Row 0	scores[0][0]	scores[0][1]	scores[0][2]
Row 1	scores[1][0]	scores[1][1]	scores[1][2]
Row 2	scores[2][0]	scores[2][1]	scores[2][2]

Tuples

- UNCHANGABLE Sequences of Data
- Syntax:

```
tuple_name = (item1, item2)
```

- Enclosed in parentheses:

```
Example: tuple1 = (“This”, “is”, “a”, “tuple”)
```

- They have elements which are indexed starting at 0
- A tuple with a single element ***must*** have a comma inside the parentheses:
    - **a = (11,)**

Operations

- ✓ Tuples support operations as lists
  - – Subscript indexing for retrieving elements
  - – Methods such as index
  - – Built in functions such as len, min, max
  - – Slicing expressions
  - – The in, +, and * operators
- ✓ Tuples do not support the methods:
  - – Append, remove, insert, reverse, sort

Basic Operations

Suppose

```
>>> tup1=(1,2,3)
>>>>tup2=(4,5,6)
>>>tup3=(‘Hi’,)
```

Python Expression	Results	Description
len(tup1)	3	Length
tup1 + tup2	(1, 2, 3, 4, 5, 6)	Concatenation
tup3* 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in tup1	True	Membership
for x in tup1: print x,	1 2 3	Iteration

Indexing, Slicing

Suppose
>>> L=(‘welcome’ , ’to’ , ’python’)

Python Expression	Results	Description
L[2]	‘python’	Offsets start at zero
L[-2]	‘to'	Negative: count from the right
L[1:]	[‘to', ‘python']	Slicing fetches sections

Same as String

Tuples and Assignment

- We can also put a tuple on the left hand side of an assignment statement
- We can even omit the parenthesis

```
(x,y) = (4,'fred')
print(x)

4
```

Tuples are Comparable

- The comparison operators work with tuples and other sequences If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
print((0,1,2)<(4,0,1))

True
```

```
print(('jones','sam')<('jones','saria’))

True
```

Deleting Tuple

```
mytuple = (1,2,3)
del mytuple
print(mytuple)
```

```
Traceback (most recent call last):
  File "C:/Users/saria/PycharmProjects/myexercises/testaki.py", line 13, in <module>
    print(mytuple)
NameError: name 'mytuple' is not defined
```

As tuple is already deleted

Tuples are more efficient than lists

- We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
- Since tuple are immutable, iterating through tuple is faster than with list.
- So there is a slight performance boost. They are more simpler in nature.
- Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

SETS

Sets

Set: object that stores a collection of data in same way as mathematical set

- All items must be unique
- Set is unordered
- Elements can be of different data types
- Sets are mutable and it contains immutable elements

e.g.:

```
>>> aset = {11, 22, 33}
>>> bset = aset
>>> aset = aset | {55}
>>> aset        {33, 11, 22, 55}
>>> bset        {33, 11, 22}
```

Creating a Set

- For empty set, call set()
- For non-empty set, call set(*argument*) where argument is an object that contains iterable elements
  - {'Alice', 'Bob', 'Carol'}
  - {'Dean'} is a singleton
- *argument* can be a list, string, or tuple
- If *argument* is a string, each character becomes a set element
- If *argument* contains duplicates, only one of the duplicates will appear in the set

Len, add, update function

- len function: returns the number of elements in the set
  - >>> myset = {'Apples', 'Bananas', 'Oranges'}
  - >>> len(myset) → 3
- add method: adds an element to a set
  - >>> myset.add(‘strawberry’)
  - >>> print(myset) -> {'Apples', 'Bananas', 'Oranges’, ‘strawberry’}
- update method: adds a group of elements to a set
    - >>> myset.update(‘1’,’2’)
    - >>> print(myset) -> {'Apples', 'Bananas', 'Oranges’,’1’,’2’}

Deleting Elements From a Set

- Remove and discard methods: remove the specified item from the set
  - The item that should be removed is passed to both methods as an argument
  - Behave differently when the specified item is not found in the set
    - remove method raises a KeyError exception
    - discard method does not raise an exception

```
>>> myset.remove(‘Apples’)
>>> print(myset)   -> {'Bananas', 'Oranges’)}
```

- clear method: clears all the elements of the set

For Loop, in, and not in Operators With a Set

- A for loop can be used to iterate over elements in a set
  - General format: for *item* in *set*:
  - The loop iterates once for each element in the set

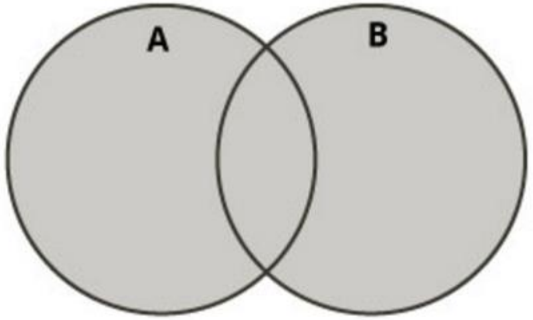
- The in operator can be used to test whether a value exists in a set
  - Similarly, the not in operator can be used to test whether a value does not exist in a set

Sets do not support indexing

```
>>> myset = {'Apples', 'Bananas', 'Oranges'}
>>> myset
            {'Bananas', 'Oranges', 'Apples'}
>>> myset[0]
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    myset[0]
TypeError: 'set' object does not support indexing
```

Boolean operations on sets (I)

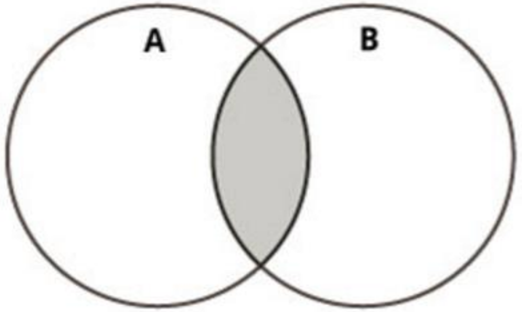
Union of two sets



- Contains all elements that are in set A or in set B
- >>> aset = {11, 22, 33}
- >>> bset = {12, 23, 33}

- Union of two sets
  - >>> aset | bset
  - {33, 22, 23, 11, 12}

Boolean operations on sets (II)

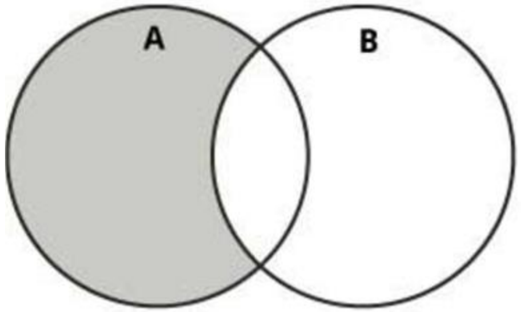


Intersection of two sets

- Contains common elements that are in both sets A and B
- >>> aset = {11, 22, 33}
- >>> bset = {12, 23, 33}

- Intersection of two sets:
  - >>> aset & bset
  - {33}

Boolean operations on sets (III)



Difference of two sets

- Contains all elements that are in A but not in B
- >>> aset = {11, 22, 33}
- >>> bset = {12, 23, 33}

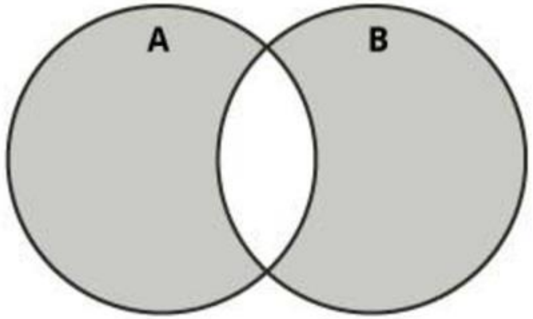
- Difference:
  - >>> aset – bset
  - {11, 22}

Boolean operations on sets (IV)

Symmetric difference of two sets

- Contains all elements that are either
  - in set A but not in set B or
  - in set B but not in set A
- >>> aset = {11, 22, 33}
- >>> bset = {12, 23, 33}

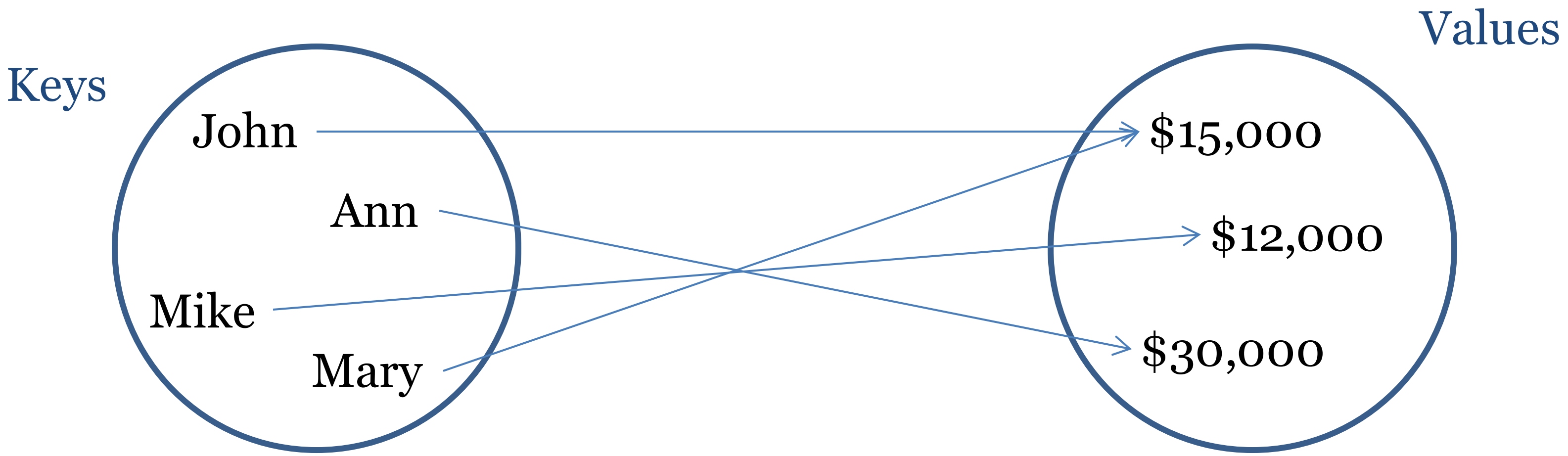
- Symmetric difference:
  - >>> aset ^ bset
  - {11, 12, 22, 23}



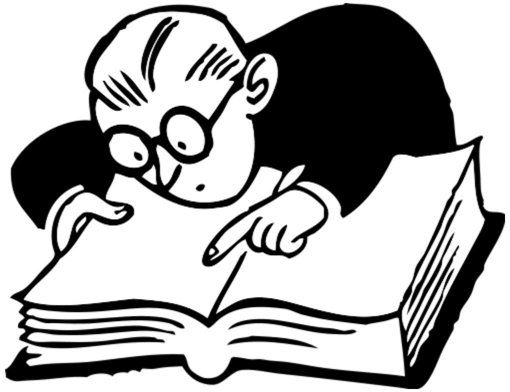
DICTIONARIES

Dictionaries

- A dictionary is a container that keeps associations between *keys* and *values*.
- Keys are unique, but a value may be associated with several keys.



Creating Dictionaries



- Each key/value pair is separated by a colon.
- You enclose the key/value pairs in braces.

- To create an empty dictionary:
  - Use {}
  - Use built-in function dict()

- Use a for loop to iterate over a dictionary
  - General format: for *key* in *dictionary*:

Sets and dictionaries

- When the braces contain key/value pairs, they denote a dictionary, not a set.
-
- The only ambiguous case is an empty {}.
- By convention, it denotes an empty dictionary, not an empty set.
-
- You can create a duplicate copy of a dictionary using the dict function:

```
oldSalaries = dict(salaries)
```

```
salaries = { “John": 15000, “Ann": 30000, “Mike": 12000, “Mary": 15000 }
```

Accessing Dictionary Values

- The subscript operator [] is used to return the value associated with a key.

- The statement:

  print(“Ann’s salary is", salaries[“Ann"])

  prints 30000

- Note that the dictionary is not a sequence-type container like a list.
- Even though the subscript operator is used with a dictionary, you cannot access the items by index or position.
- A value can only be accessed using its associated key

Searching For Keys

- The key supplied to the subscript operator must be a valid key in the dictionary or a KeyError exception will be raised.
- To find out whether a key is present in the dictionary, use the **in** (or **not in**) operator:

```
if “Ann" in salaries :
   print(“Ann’s salary is", salaries[“Ann”])
else :
   print(“Ann’s salary is not in my list.”)
```

Dictionaries are mutable

- age = {'Alice' : 25, 'Bob' : 28}
- saved = age
- age['Bob'] = 29
- age
- {'Bob': 29, 'Alice': 25}
- saved
- {'Bob': 29, 'Alice': 25}

Adding Elements to an Existing Dictionary

To add a new key-value pair:

dictionary[*key*] = *value*

- If key exists in the dictionary, the value associated with it will be changed

Updating Dictionaries

```
>>> age = {'Alice': 26 , 'Carol' : 22}

>>> age.update({'Bob' : 29})
>>> age
{'Bob': 29, 'Carol': 22, 'Alice': 26}

>>> age.update({'Carol' : 23})
>>> age
{'Bob': 29, 'Carol': 23, 'Alice': 26}
```

Returning a value

- `>>> age = {'Bob': 29, 'Carol': 23, 'Alice': 26}`
- `>>> age.get('Bob')`
- `29`

- `>>> age['Bob']`
- `29`

Displaying Contents

```
>>> age = {'Alice' : 25, 'Carol': 'twenty-two’}

>>> age.items()
dict_items([ ('Alice', 25), ('Carol', 'twenty-two')])
>>> age.keys()
dict_keys([ 'Alice', 'Carol'])
 >>> age.values()
dict_values([25, 'twenty-two'])
```

Removing a specific item

- >>> a = {'Alice' : 26, 'Carol' : 'twenty-two’}

- >>> a
- {'Carol': 'twenty-two', 'Alice': 26}

- >>> a.pop('Carol’)
- 'twenty-two’

- >>> a
- {'Alice': 26}

Remove a random item

- >>> age = {'Bob': 29, 'Carol': 23, 'Alice': 26}
- >>> age.popitem()
- ('Bob', 29)
- >>> age
- {'Carol': 23, 'Alice': 26}

- >>> age.popitem()
- ('Carol', 23)
- >>> age
- {'Alice': 26}

Number of Elements and Mixing Data Types

- len function: used to obtain number of elements in a dictionary
- clear method: deletes all the elements in a dictionary, leaving it empty
  - Format: *dictionary*.clear()
- Values stored in a single dictionary can be of different types

Different Ways Of Doing The Same Thing

- **Lists**

```
prerequisites = ["COP 2271c", "Introduction to Computation and Programming", 3]
print(prerequisites[2])  # Prints the element at index 2
```

- **Sets**

```
cheesePizza = {"Creamy garlic", "Parmesan sauce", "Cheese", "Toasted Parmesan"}
if "Toasted Parmesan" in cheesePizza:
```

- **Dictionaries**

```
salaries = {"John": 15000, "Ann": 30000, "Mike": 12000, "Mary": 15000 }
print("Ann’s salary  is", salaries["Ann"])
```

Function

- Function is defined using the def keyword
- Creating a Function
  - def my_function():
    print("Hello from a function")

- Calling a Function
  - **my_function()**

- Parameters
  - def my_function(**fname**):
    print(fname + " Refsnes")

  - my_function(**"Emil"**)

Function

- Default Parameter Value

  -
    ```
    def my_function(country = "Norway"):
      print("I am from " + country)

    my_function("Sweden")
    my_function()
    ```

- Return Values

  -
    ```
    def my_function(x):
      return 5 * x

    print(my_function(3))
    ```

Use case: Car Rental

Use case: Car Rental

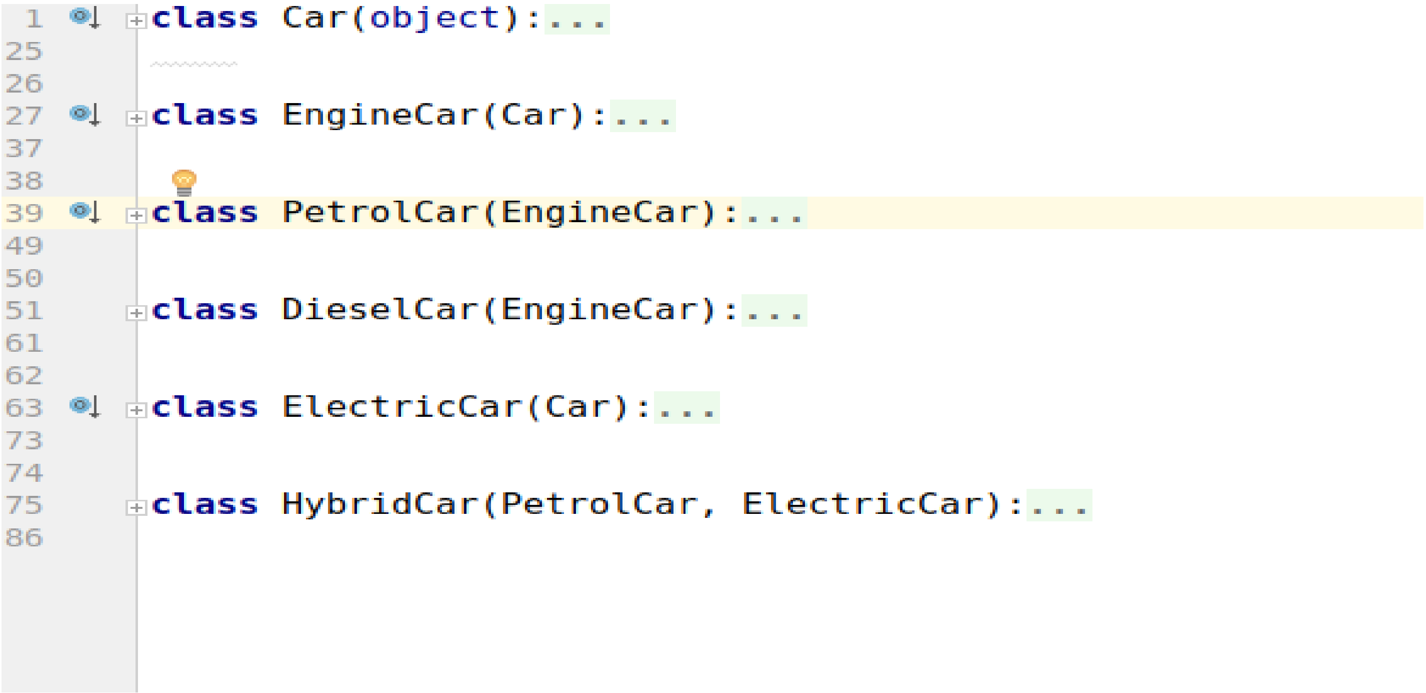
- The goal of this use case is to calculate the money one should pay when renting a car.

- Surely this money will be affected by the kind of car, distance, kind of petrol, number of days…

- So in this use case, there is a couple of questions that a user answers then the amount of money one should pay will be printed.

Module cars

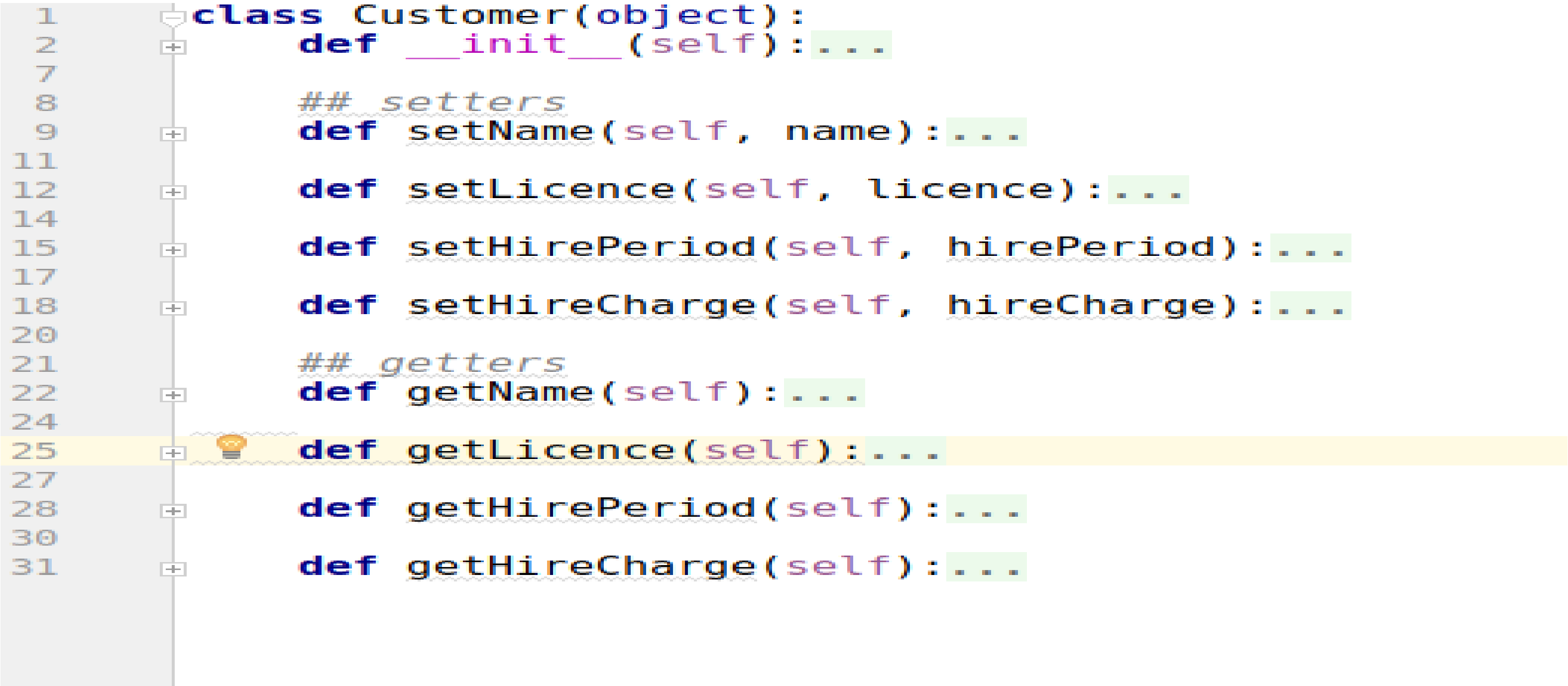
- It contains six classes which helps to choose various features of a car for hiring



```
class Car(object):...


class EngineCar(Car):...


class PetrolCar(EngineCar):...


class DieselCar(EngineCar):...


class ElectricCar(Car):...


class HybridCar(PetrolCar, ElectricCar):...

```

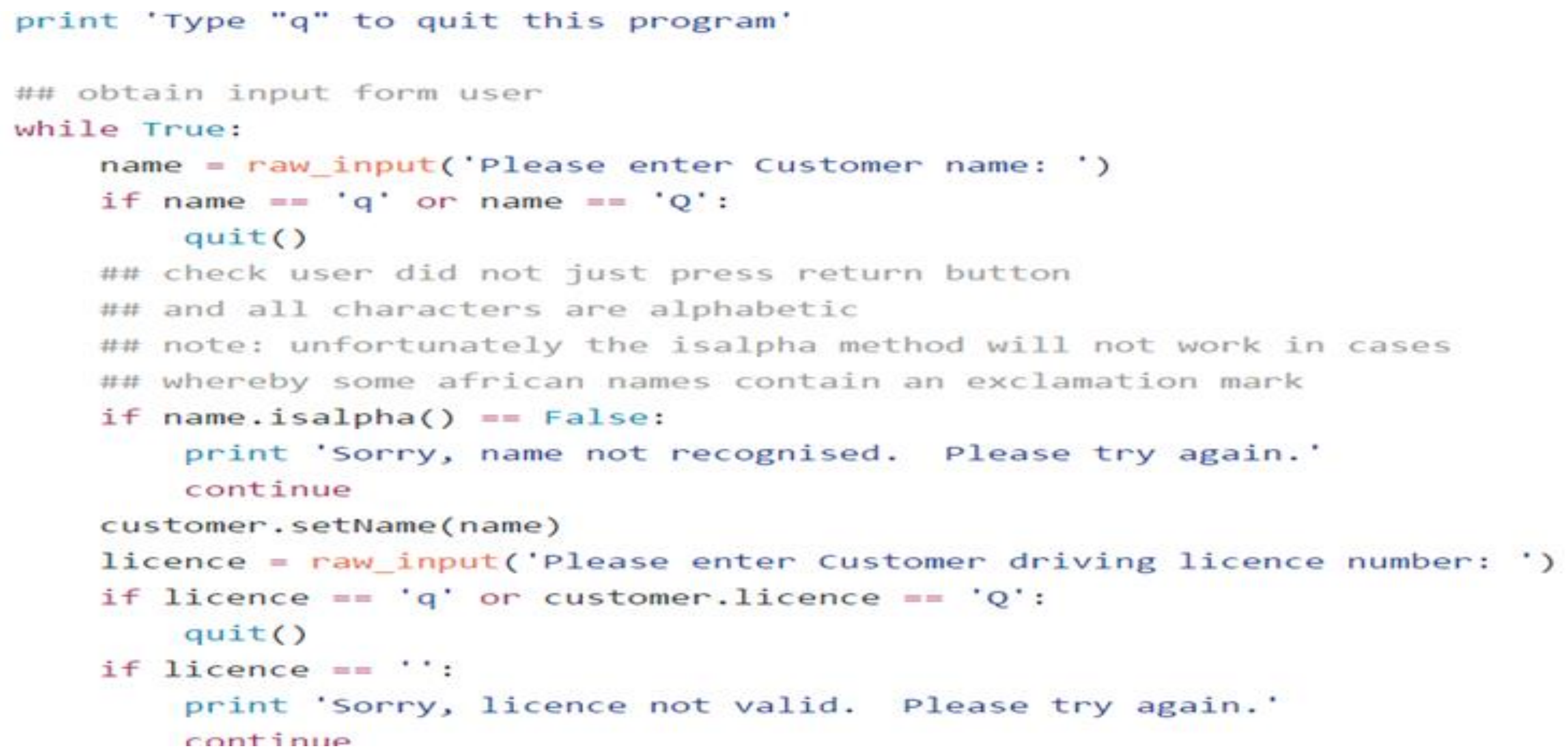
Module customer

- This module includes details about information of the customer



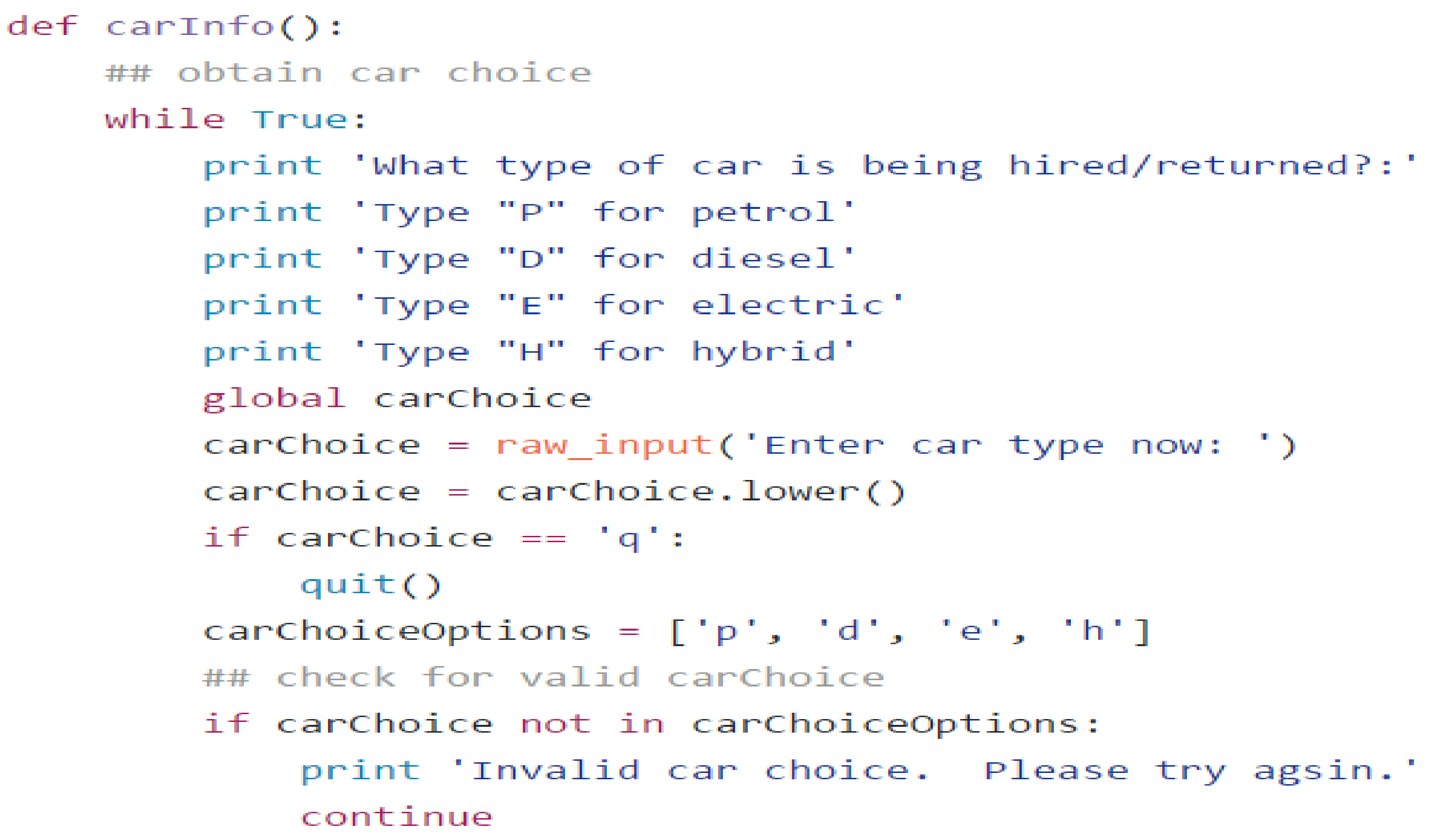
```python
class Customer(object):
    def __init__(self):...

    ## setters
    def setName(self, name):...

    def setLicence(self, licence):...

    def setHirePeriod(self, hirePeriod):...

    def setHireCharge(self, hireCharge):...

    ## getters
    def getName(self):...

    def getLicence(self):...

    def getHirePeriod(self):...

    def getHireCharge(self):...
```

Getting information of the customer



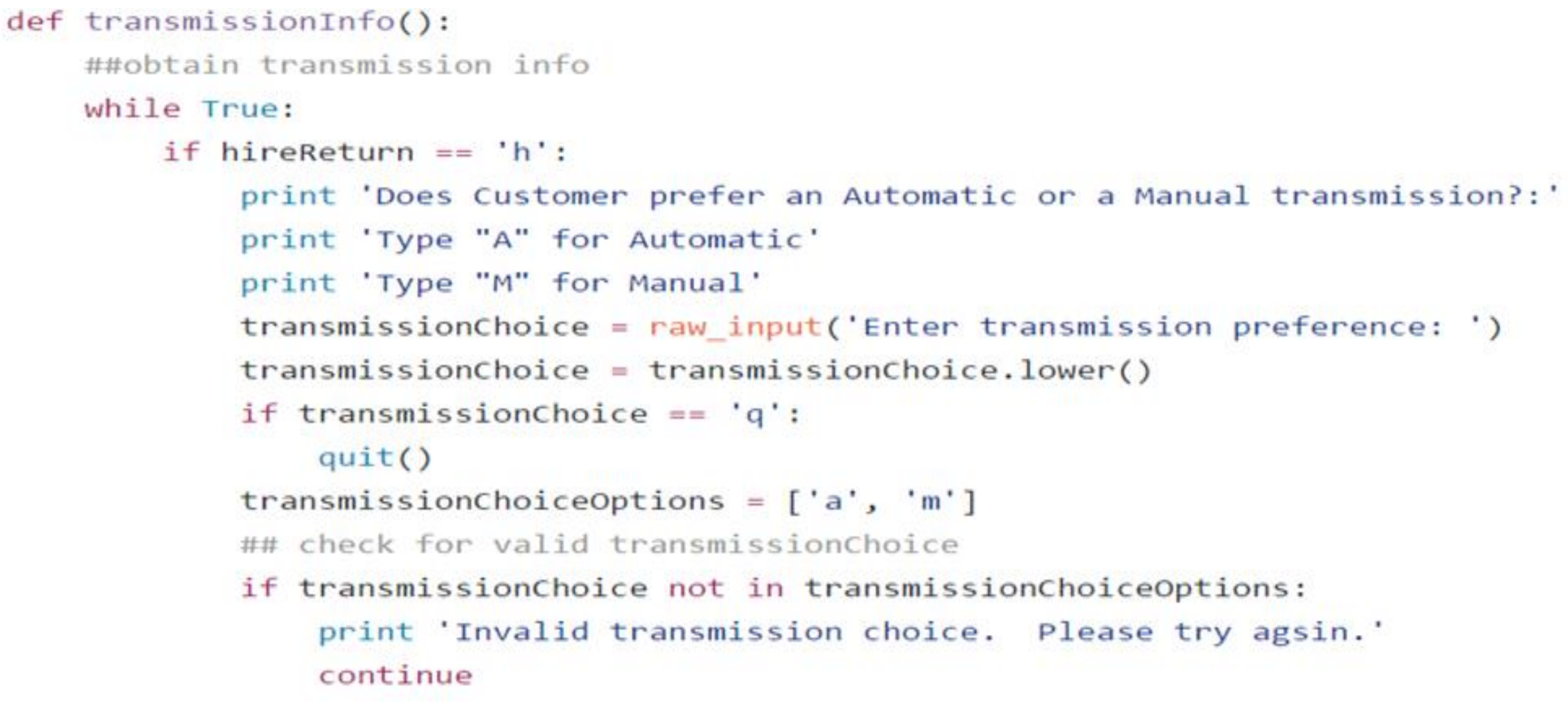
```
print 'Type "q" to quit this program'

## obtain input form user
while True:
    name = raw_input('Please enter Customer name: ')
    if name == 'q' or name == 'Q':
        quit()
    ## check user did not just press return button
    ## and all characters are alphabetic
    ## note: unfortunately the isalpha method will not work in cases
    ## whereby some african names contain an exclamation mark
    if name.isalpha() == False:
        print 'Sorry, name not recognised.  Please try again.'
        continue
    customer.setName(name)
    licence = raw_input('Please enter Customer driving licence number: ')
    if licence == 'q' or customer.licence == 'Q':
        quit()
    if licence == '':
        print 'Sorry, licence not valid.  Please try again.'
        continue
```

Choosing one of the car options available



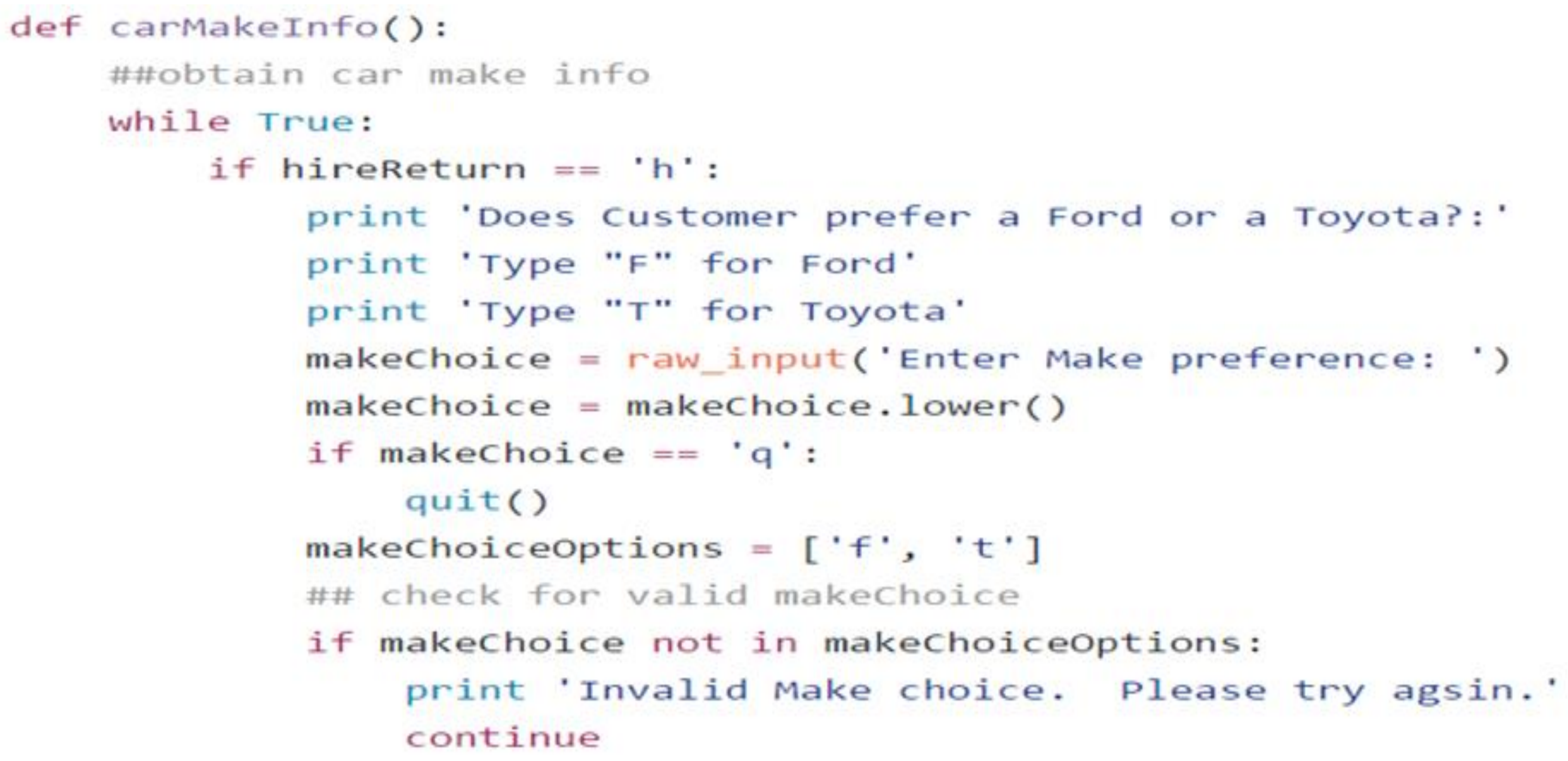
```
def carInfo():
    ## obtain car choice
    while True:
        print 'What type of car is being hired/returned?:'
        print 'Type "P" for petrol'
        print 'Type "D" for diesel'
        print 'Type "E" for electric'
        print 'Type "H" for hybrid'
        global carChoice
        carChoice = raw_input('Enter car type now: ')
        carChoice = carChoice.lower()
        if carChoice == 'q':
            quit()
        carChoiceOptions = ['p', 'd', 'e', 'h']
        ## check for valid carChoice
        if carChoice not in carChoiceOptions:
            print 'Invalid car choice.  Please try agsin.'
            continue
```

Automatic or Manual car



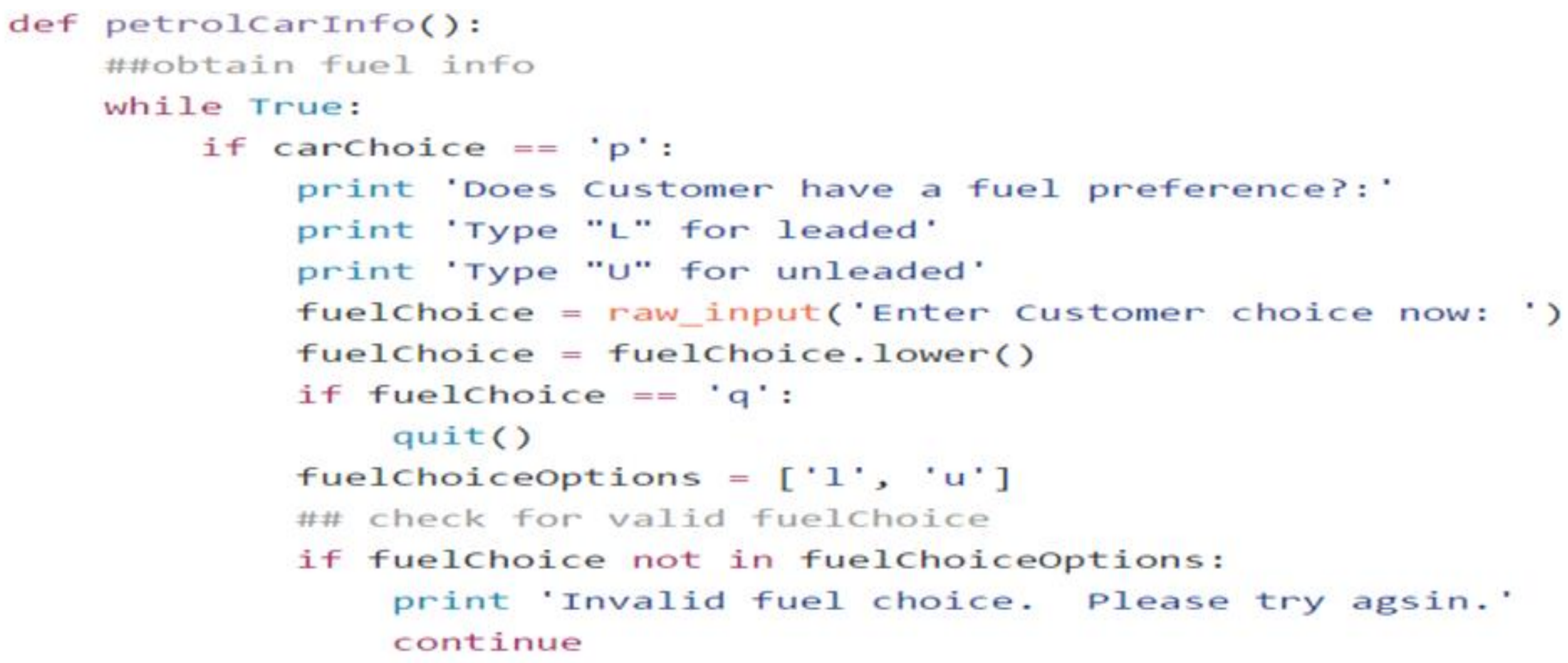
```
def transmissionInfo():
    ##obtain transmission info
    while True:
        if hireReturn == 'h':
            print 'Does Customer prefer an Automatic or a Manual transmission?:'
            print 'Type "A" for Automatic'
            print 'Type "M" for Manual'
            transmissionChoice = raw_input('Enter transmission preference: ')
            transmissionChoice = transmissionChoice.lower()
            if transmissionChoice == 'q':
                quit()
            transmissionChoiceOptions = ['a', 'm']
            ## check for valid transmissionChoice
            if transmissionChoice not in transmissionChoiceOptions:
                print 'Invalid transmission choice.  Please try agsin.'
                continue
```

Ford Or Toyota



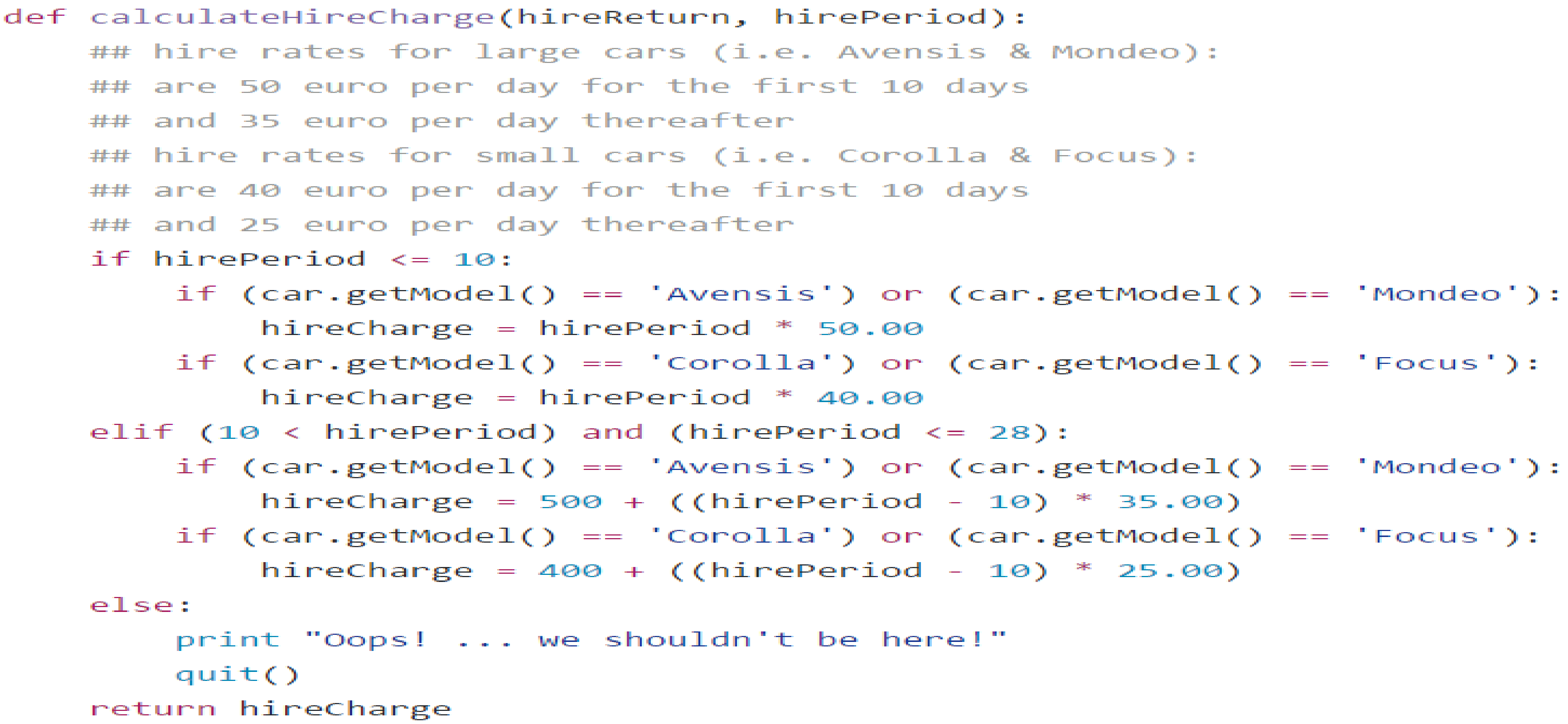
```
def carMakeInfo():
    ##obtain car make info
    while True:
        if hireReturn == 'h':
            print 'Does Customer prefer a Ford or a Toyota?:'
            print 'Type "F" for Ford'
            print 'Type "T" for Toyota'
            makeChoice = raw_input('Enter Make preference: ')
            makeChoice = makeChoice.lower()
            if makeChoice == 'q':
                quit()
            makeChoiceOptions = ['f', 't']
            ## check for valid makeChoice
            if makeChoice not in makeChoiceOptions:
                print 'Invalid Make choice.  Please try agsin.'
                continue
```

Fuel Preference



```
def petrolCarInfo():
    ##obtain fuel info
    while True:
        if carChoice == 'p':
            print 'Does Customer have a fuel preference?:'
            print 'Type "L" for leaded'
            print 'Type "U" for unleaded'
            fuelChoice = raw_input('Enter Customer choice now: ')
            fuelChoice = fuelChoice.lower()
            if fuelChoice == 'q':
                quit()
            fuelChoiceOptions = ['l', 'u']
            ## check for valid fuelChoice
            if fuelChoice not in fuelChoiceOptions:
                print 'Invalid fuel choice.  Please try agsin.'
                continue
```

Rates of hiring



```
def calculateHireCharge(hireReturn, hirePeriod):
    ## hire rates for large cars (i.e. Avensis & Mondeo):
    ## are 50 euro per day for the first 10 days
    ## and 35 euro per day thereafter
    ## hire rates for small cars (i.e. Corolla & Focus):
    ## are 40 euro per day for the first 10 days
    ## and 25 euro per day thereafter
    if hirePeriod <= 10:
        if (car.getModel() == 'Avensis') or (car.getModel() == 'Mondeo'):
            hireCharge = hirePeriod * 50.00
        if (car.getModel() == 'Corolla') or (car.getModel() == 'Focus'):
            hireCharge = hirePeriod * 40.00
    elif (10 < hirePeriod) and (hirePeriod <= 28):
        if (car.getModel() == 'Avensis') or (car.getModel() == 'Mondeo'):
            hireCharge = 500 + ((hirePeriod - 10) * 35.00)
        if (car.getModel() == 'Corolla') or (car.getModel() == 'Focus'):
            hireCharge = 400 + ((hirePeriod - 10) * 25.00)
    else:
        print "Oops! ... we shouldn't be here!"
        quit()
    return hireCharge
```

References

- https://github.com/wade12/ProgrammingForBigDataCA2CarRental/blob/master/carRentalApp.py

- http://www.w3resource.com/python-exercises/python-conditional-statements-and-loop-exercises.php

- https://www.learnpython.org/

Thank you