Logbook Exercise 1

Insert a 'code' cell below. In this do the following:

- line 1 - create a list named "shopping_list" with items: milk, eggs, bread, cheese, tea, coffee, rice, pasta, milk, tea (NOTE: the duplicate items are intentional)
- line 2 - print the list along with a message e.g. "This is my shopping list ..."
- line 3 - create a tuple named "shopping_tuple" with the same items
- line 4 - print the tuple with similar message e.g. "This is my shopping tuple ..."
- line 5 - create a set named "shopping_set" from "shopping_list" by using the set() method
- line 6 - print the set with appropriate message and check duplicate items have been removed
- line 7 - make a dictionary "shopping_dict" - copy and paste the following items and prices: "milk": "£1.20", "eggs": "£0.87", "bread": "£0.64", "cheese": "£1.75", "tea": "£1.06", "coffee": "£2.15", "rice": "£1.60", "pasta": "£1.53".
- line 8 - print the dictionary with an appropriate message

An example of fully described printed output is presented below (some clues here also) Don't worry of your text output is different - it is the contents of the compund variables that matter

```
This is my shopping list ['milk', 'eggs', 'bread', 'cheese',
'tea', 'coffee', 'rice', 'pasta', 'milk', 'tea']
This is my shopping tuple ('milk', 'eggs', 'bread', 'cheese',
'tea', 'coffee', 'rice', 'pasta', 'milk', 'tea')
This is my Shopping_set with duplicates removes {'rice', 'milk',
'pasta', 'cheese', 'eggs', 'tea', 'bread', 'coffee'}
This is my shopping_dict {'milk': '£1.20', 'eggs': '£0.87',
'bread': '£0.64', 'cheese': '£1.75', 'tea': '£1.06', 'coffee':
'£2.15', 'rice': '£1.60', 'pasta': '£1.53'}
```

In [1]:
```
# List named "shopping_list" with given items
shopping_list = ["milk", "eggs", "bread", "cheese", "tea", "coffee", "rice", "pa
# Print the shopping list
print("This is my shopping list:", shopping_list)

# Creating a tuple named "shopping_tuple" with the same items as in the list
shopping_tuple = tuple(shopping_list)
# Print the shopping tuple
print("This is my shopping tuple:", shopping_tuple)

# Creating a set named "shopping_set" from "shopping_list" to remove duplicates
shopping_set = set(shopping_list)
# Printing the set without duplicate items
print("This is my Shopping_set with duplicates removed:", shopping_set)

# Creating a dictionary named "shopping_dict" with items and their prices
shopping_dict = {
    "milk": "£1.20",
    "eggs": "£0.87",
    "bread": "£0.64",
    "cheese": "£1.75",
    "tea": "£1.06",
    "coffee": "£2.15",
    "rice": "£1.60",
    "pasta": "£1.53"
}
# Printing the dictionary with an appropriate message
print("This is my shopping_dict:", shopping_dict)
```

```
This is my shopping list: ['milk', 'eggs', 'bread', 'cheese', 'tea', 'coffee', '
rice', 'pasta', 'milk', 'tea']
This is my shopping tuple: ('milk', 'eggs', 'bread', 'cheese', 'tea', 'coffee',
'rice', 'pasta', 'milk', 'tea')
This is my Shopping_set with duplicates removed: {'bread', 'tea', 'eggs', 'rice'
, 'milk', 'cheese', 'coffee', 'pasta'}
This is my shopping_dict: {'milk': '£1.20', 'eggs': '£0.87', 'bread': '£0.64', '
cheese': '£1.75', 'tea': '£1.06', 'coffee': '£2.15', 'rice': '£1.60', 'pasta':
'£1.53'}
```

Logbook Exercise 2

Create a 'code' cell below. In this do the following:

```
line 1 - Use a comment to title your exercise - e.g. "Unit 2
Exercise"
line 2 - create a list ... li = ["USA","Mexico","Canada"]
line 3 - append "Greenland" to the list
l4 - print the list to de,monstrate that Greenland is attached
l5 - remove "Greenland"
l6 - print the list to de,monstrate that Greenland is removed
l7 - insert "Greenland" at the beginning of the list
l8 - print the resul of l7
l9 - shorthand slice the list to extract the first two items -
simultaneausly print the output
l10 - use a negative index to extract the second to last item -
simultaneausly print the output
l11 - use a splitting sequence to extract the middle two items -
simultaneausly print the output
```

An example of fully described printed output is presented below (some clues here also) Don't worry of your text output is different - it is the contents of the list that matter

li.append('Greenland') gives ... ['USA', 'Mexico', 'Canada', 'Greenland'] li.remove('Greenland') gives ... ['USA', 'Mexico', 'Canada'] li.insert(0,'Greenland') gives ... ['Greenland', 'USA', 'Mexico', 'Canada'] li[:2] gives ... ['Greenland', 'USA'] li[-2] gives ... Mexico li[1:3] gives ... ['USA', 'Mexico']

In [2]:
```
# Unit 2 Exercise

# Creating a list
country_list = ["USA", "Mexico", "Canada"]

# Appending "Greenland" to the list
country_list.append("Greenland")

# Demonstrating that "Greenland" is attached
print("country_list.append('Greenland') gives ... ", country_list)

# Removing "Greenland" from the list
country_list.remove("Greenland")

# Demonstrating that "Greenland" is removed
print("country_list.remove('Greenland') gives ... ", country_list)

# Inserting "Greenland" at the beginning of the list
country_list.insert(0, "Greenland")

# Printing the result of the insertion
print("country_list.insert(0,'Greenland') gives ... ", country_list)

# Using shorthand slice to extract the first two items
print("country_list[:2] gives ... ", country_list[:2])

# Using a negative index to extract the second to last item
print("country_list[-2] gives ... ", country_list[-2])

# Using a slicing sequence to extract the middle two items
print("country_list[1:3] gives ... ", country_list[1:3])
```

```
country_list.append('Greenland') gives ...  ['USA', 'Mexico', 'Canada', 'Greenla
nd']
country_list.remove('Greenland') gives ...  ['USA', 'Mexico', 'Canada']
country_list.insert(0,'Greenland') gives ...  ['Greenland', 'USA', 'Mexico', 'Ca
nada']
country_list[:2] gives ...  ['Greenland', 'USA']
country_list[-2] gives ...  Mexico
country_list[1:3] gives ...  ['USA', 'Mexico']
```

Logbook Exercise 3

Create a 'code' cell below. In this do the following:

```
on the first line create the following set ...
a=[0,1,2,3,4,5,6,7,8,9,10]
on the second line create the following set ...
b=[0,5,10,15,20,25]
on the third line create the following dictionary ...
topscores={"Jo":999, "Sue":987, "Tara":960; "Mike":870}
use a combination of print() and type() methods to produce the
following output
```

list a is ... [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] list b is ... [0, 5, 10, 15, 20, 25] The type of a is now ... <class 'list'>

```
on the next 2 lines convert list a and b to sets using set()
on the following lines use a combination of print(), type() and
set notaion (e.g. 'a & b', 'a | b', 'b-a') to obtain the
following output
```

set a is ... {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10} set b is ... {0, 5, 10, 15, 20, 25} The type of a is now ... <class 'set'> Intersect of a and b is [0, 10, 5] Union of a and b is [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25] Items unique to set b are {25, 20, 15}

```
on the next 2 lines use print(), '.keys()' and '.values()'
methods to obtain the following output
```

topscores dictionary keys are dict_keys(['Jo', 'Sue', 'Tara', 'Mike']) topscores dictionary values are dict_values([999, 987, 960, 870])

In [3]:
```python
# Create a list named 'a'
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Create a list named 'b'
b = [0, 5, 10, 15, 20, 25]

# Create a dictionary named 'topscores'
topscores = {"Jo": 999, "Sue": 987, "Tara": 960, "Mike": 870}

# Using print() and type() methods
print("list a is ... ", a)
print("list b is ... ", b)
print("The type of a is now ... ", type(a))

# Converting lists a and b to sets
a = set(a)
b = set(b)

# Using print(), type() and set notation
print("set a is ... ", a)
print("set b is ... ", b)
print("The type of a is now ... ", type(a))
print("Intersect of a and b is", sorted(list(a & b)))
print("Union of a and b is", sorted(list(a | b)))
print("Items unique to set b are", b - a)

# Using print(), '.keys()' and '.values()' methods
print("topscores dictionary keys are", topscores.keys())
print("topscores dictionary values are", topscores.values())
```

```
list a is ...  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
list b is ...  [0, 5, 10, 15, 20, 25]
The type of a is now ...  <class 'list'>
set a is ...  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
set b is ...  {0, 5, 10, 15, 20, 25}
The type of a is now ...  <class 'set'>
Intersect of a and b is [0, 5, 10]
Union of a and b is [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25]
Items unique to set b are {25, 20, 15}
topscores dictionary keys are dict_keys(['Jo', 'Sue', 'Tara', 'Mike'])
topscores dictionary values are dict_values([999, 987, 960, 870])
```

Logbook Exercise 4

Create a 'code' cell below. In this do the following:

```
Given the following 4 lists of names, house number and street
addresses, towns and postcodes ...
```

["T Cruise","D Francis","C White"] ["2 West St","65 Deadend Cls","15 Magdalen Rd"] ["Canterbury", "Reading", "Oxford"] ["CT8 23RD", "RG4 1FG", "OX4 3AS"]

```
write a Custom 'address_machine' function that formats 'name',
'hs_number_street', 'town', 'postcode' with commas and spaces
between items
create a 'newlist' that repeatedly calls 'address_machine' and
'zips' items from the 4 lists
write a 'for loop' that iterates over 'new list' and prints each
name and address on a separate line
the output should appear as follows
```

T Cruise, 2 West St, Canterbury, CT8 23RD D Francis, 65 Deadend Cls, Reading, RG4 1FG C White, 15 Magdalen Rd, Oxford, OX4 3AS

```
HINT: look at "# CUSTOM FUNCTION WORKED EXAMPLES 3 & 4" above
```

In [4]:
```python
# Lists of names, addresses, towns, and postcodes
names = ["T Cruise", "D Francis", "C White"]
hs_number_street = ["2 West St", "65 Deadend Cls", "15 Magdalen Rd"]
towns = ["Canterbury", "Reading", "Oxford"]
postcodes = ["CT8 23RD", "RG4 1FG", "OX4 3AS"]

# Custom function to format address
def address_machine(name, hs_number_street, town, postcode):
    return f"{name}, {hs_number_street}, {town}, {postcode}"

# Using a list comprehension to call 'address_machine' directly with zipped list
formatted_addresses = [address_machine(name, hs, town, pc) for name, hs, town, p

# Printing each formatted address
for address in formatted_addresses:
    print(address)
```

```
T Cruise, 2 West St, Canterbury, CT8 23RD
D Francis, 65 Deadend Cls, Reading, RG4 1FG
C White, 15 Magdalen Rd, Oxford, OX4 3AS
```

Logbook Exercise 5

Create a 'code' cell below. In this do the following:

```
Create a super class "Person" that takes three string and one
integer parameters for first and second name, UK Postcode and
age in years.
Give "Person" a method "greeting" that prints a statement along
the lines "Hello, my name is Freddy Jones. I am 22 years old and
my postcode is HP6 7AJ"
Create a "Student" class that extends/inherits "Person" and
takes additional parameters for degree_subject and student_ID.
give "Student" a "studentGreeting" method that prints a
statement along the lines "My student ID is SN123456 and I am
reading Computer Science"
Use either Python {} format or C-type %s/%d notation to format
output strings
Create 3 student objects and persist these in a list
Iterate over the three objects and call their "greeting" and
"studentGreeting" methods
Output should be along the lines of the following
```

Hello, my name is Dick Turpin. I am 32 years old and my postcode is HP11 2JZ My student ID is DT123456 and I am reading Highway Robbery

Hello, my name is Dorothy Turpin. I am 32 years old and my postcode is SO14 7AA My student ID is DT123457 and I am reading Law

Hello, my name is Oliver Cromwell. I am 32 years old and my postcode is OX35 14RE My student ID is OC123456 and I am reading History

In [5]:

```python
# Super class "Person"
class Person:
    def __init__(self, first_name, second_name, postcode, age):
        self.first_name = first_name
        self.second_name = second_name
        self.postcode = postcode
        self.age = age

    def greeting(self):
        print("Hello, my name is %s %s. I am %d years old and my postcode is %s"

# "Student" class that extends "Person"
class Student(Person):
    def __init__(self, first_name, second_name, postcode, age, degree_subject, s
        super().__init__(first_name, second_name, postcode, age)
        self.degree_subject = degree_subject
        self.student_ID = student_ID

    def studentGreeting(self):
        print("My student ID is %s and I am reading %s" % (self.student_ID, self

# Creating 3 student objects
students = [
    Student("Dick", "Turpin", "HP11 2JZ", 32, "Highway Robbery", "DT123456"),
    Student("Dorothy", "Turpin", "SO14 7AA", 32, "Law", "DT123457"),
    Student("Oliver", "Cromwell", "OX35 14RE", 32, "History", "OC123456")
]

# Iterating over the student objects and calling their "greeting" and "studentGr
for student in students:
    student.greeting()
    student.studentGreeting()
    print("")
```

```
Hello, my name is Dick Turpin. I am 32 years old and my postcode is HP11 2JZ
My student ID is DT123456 and I am reading Highway Robbery

Hello, my name is Dorothy Turpin. I am 32 years old and my postcode is SO14 7AA
My student ID is DT123457 and I am reading Law

Hello, my name is Oliver Cromwell. I am 32 years old and my postcode is OX35 14R
E
My student ID is OC123456 and I am reading History
```

Logbook Exercise 6

- Examine Steve Lipton's "simplest ever" version for the MVC
- Note how when the MyController object is initialised it:
  - passes a reference to itself to the MyModel and MyView objects it creates
  - thereby allowing MyModel and MyView to create 'delegate' vc (virtual control) aliases to call back the MyController object
- When you feel you have understood MVC messaging and delegation add code for a new button that removes items from the list
- The end result should be capable of creating the output below
- Clearly ***comment your code*** to highlight the insertions you have made
- Note: if you don't see the GUI immediately look for the icon Jupyter icon in your task bar (also higlighted below)

In [19]:
```
#MVC_Template_01
#2014 May 23  by Steven Lipton http://makeAppPie.com
#Controller initializing MVC -- simplest version possible.
#Additional annotations by R Mather - Oct 2018

# MVC EXTENDED WITH DELETE BUTTON

# Tkinter is a Python binding to the Tk GUI toolkit/library - RM
from tkinter import *

# A Model-View-Controller framework for TKinter.
# Model: Data Structure. Controller can send messages to it, and model can respo
# View : User interface elements. Controller can send messages to it. View can c
# Controller: Ties View and Model together. turns UI responses into chages in da



#Controller: Ties View and Model together.
#       --Performs actions based on View events.
#       --Sends messages to Model and View and gets responses
#       --Has Delegates - RM NOTE: essential for communications between view and

class MyController():
    def __init__(self,parent):
        self.parent = parent
        # CRITICAL PART OF THE MVC
        # RM NOTES:
        # MyController passes a reference to itself (self) when it creates MyMod
        # This allows the MyModel and MyView objects to create 'delegates' (see
        # The vc objects can then call methods on the MyControl object, thereby
        self.model = MyModel(self)    # initializes the model
        self.view = MyView(self)  #initializes the view
        # Initialize objects in view
        self.view.setEntry_text('Add to Label') # a non cheat way to do MVC with
        self.view.setLabel_text('Ready')

    #event handlers
    def quitButtonPressed(self):
        self.parent.destroy()
    def addButtonPressed(self):
        self.view.setLabel_text(self.view.entry_text.get())
        # CRITICAL PART OF THE MVC
        # RM NOTES - In one operation:
        #[1] Get the text from the VIEW;
        #[2] Add it to the MODEL by appending it to the list
        self.model.addToList(self.view.entry_text.get())

    def listChangedDelegate(self):
        #model internally changes and needs to signal a change
        print(self.model.getList())

    # Event handler
    def removeButtonPressed(self):
        # Remove the last item from the model's list
        if self.model.getList():
            self.model.removeFromList()
            self.view.setLabel_text("Last item removed")
        else:
            self.view.setLabel_text("List is empty")

#View : User interface elements.
#       --Controller can send messages to it.
#       --View can call methods from Controller vc when an event happens
#       --View can call methods from controller vc when an event happens.
#       --NEVER communicates with Model.
#       --Has setters and getters to communicate with controller

class MyView(Frame):

    def __init__(self,vc):
        self.frame=Frame()
        self.frame.grid(row = 0,column=0)
        self.vc = vc
        self.entry_text = StringVar()
        self.entry_text.set('nil')
        self.label_text = StringVar()
        self.label_text.set('nil')
        self.loadView()

    def loadView(self):
        quitButton = Button(self.frame, text='Quit', command=self.vc.quitButtonP
        addButton = Button(self.frame, text="Add", command=self.vc.addButtonPres
        # New remove button
        removeButton = Button(self.frame, text="Remove", command=self.vc.removeB
        entry = Entry(self.frame, textvariable=self.entry_text).grid(row=1, colu
        label = Label(self.frame, textvariable=self.label_text).grid(row=2, colu


    def getEntry_text(self):
    #returns a string of the entry text
        return self.entry_text.get()
    def setEntry_text(self,text):
    #sets the entry text given a string
        self.entry_text.set(text)
    def getLabel_text(self):
    #returns a string of the Label text
        return self.label_text.get()
    def setLabel_text(self,text):
    #sets the label text given a string
        self.label_text.set(text)

#Model: Data Structure.
#   --Controller can send messages to it, and model can respond to message.
#   --Uses delegates from vc to send messages to the Control of internal change
#   --NEVER communicates with View
#   --Has setters and getters to communicate with Controller

class MyModel():
    def __init__(self,vc):
        self.vc = vc
        self.myList = ['duck','duck','goose']
        self.count = 0
#Delegates-- Model would call this on internal change
    def listChanged(self):
        self.vc.listChangedDelegate()
#setters and getters
    def getList(self):
        return self.myList
    def initListWithList(self, aList):
        self.myList = aList
    def addToList(self,item):
        print("returned")
        myList = self.myList
        myList.append(item)
        self.myList=myList
        self.listChanged()
    # New function
```

```python
    def removeFromList(self):
        if self.myList:  # Check if the list is not empty
            # Remove the last item from the list
            removed_item = self.myList.pop()
            # Print the removed item for confirmation
            print(f"Removed: {removed_item}")
            self.listChanged()  # Notify the controller about the change


def main():
    # Create a root window from the Tk GUI class
    root = Tk()
    frame = Frame(root,bg='#0555ff' )
    root.title('Hello Penguins')
    app = MyController(root)
    root.mainloop()

if __name__ == '__main__':
    main()
```

```
returned
['duck', 'duck', 'goose', 'penguin']
returned
['duck', 'duck', 'goose', 'penguin', 'chicken']
returned
['duck', 'duck', 'goose', 'penguin', 'chicken', 'turkey']
returned
['duck', 'duck', 'goose', 'penguin', 'chicken', 'turkey', 'pheasant']
Removed: pheasant
['duck', 'duck', 'goose', 'penguin', 'chicken', 'turkey']
```

Logbook Exercise 7

- Your task is to extend the Observer example below with a pie-chart view of model data and to copy this cell and the solution to your logbook
- The bar chart provides a useful example of structure
- Partial code is provided below for insertion, completion (note '####' requires appropriate replacement) and implementation
- you will also need to create an 'observer' object from the PieView class and attach it to the first 'model'

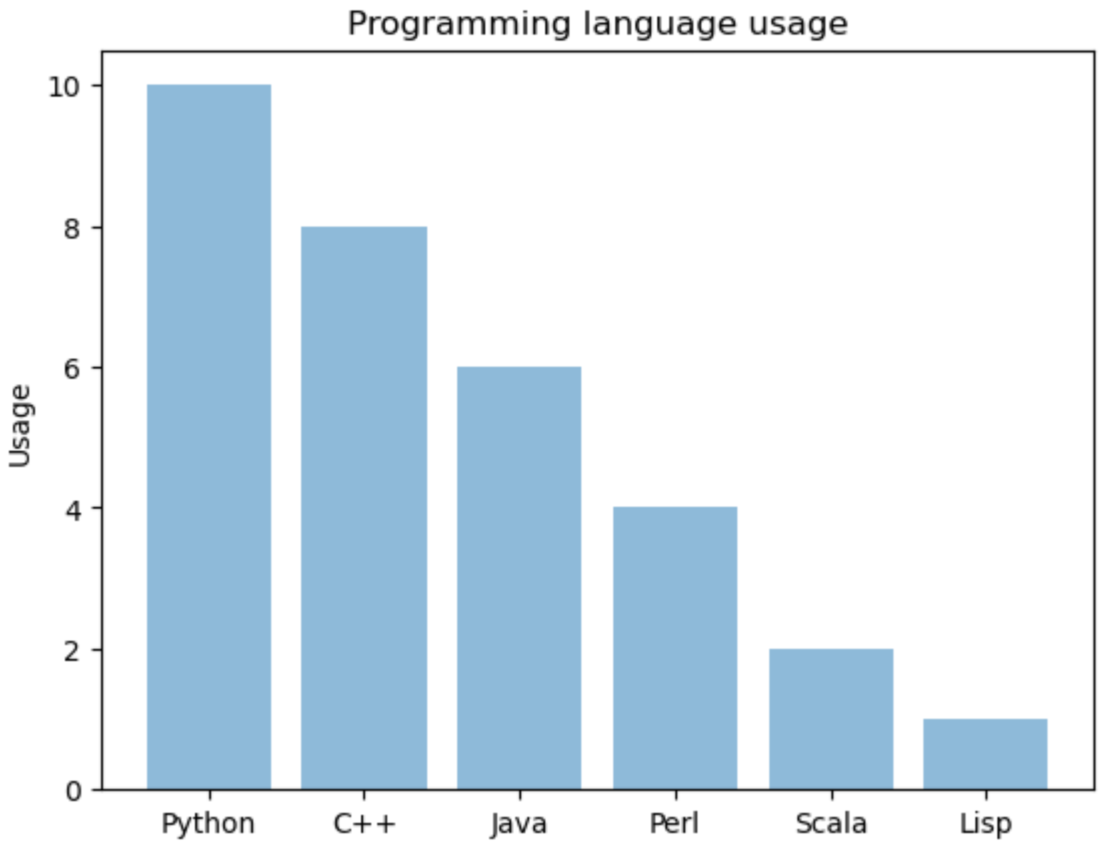
```
# Pie chart viewer/ConcreteObserver - overrides the update()
method
class PieView(####):

    def update(####, ####): #Alert method that is invoked when
the notify() method in a concrete subject is invoked
        # Pie chart, where the slices will be ordered and
plotted counter-clockwise:
        labels = ####
        sizes = ####
        explode = (0.1, 0, 0, 0, 0, 0)  # only "explode" the 1st
slice
        fig1, ax1 = plt.subplots()
        ax1.pie(sizes, explode=explode, labels=labels,
autopct='%1.1f%%', shadow=True, startangle=90)
        ax1.axis('equal')  # Equal aspect ratio ensures that pie
is drawn as a circle.
        plt.####()
```

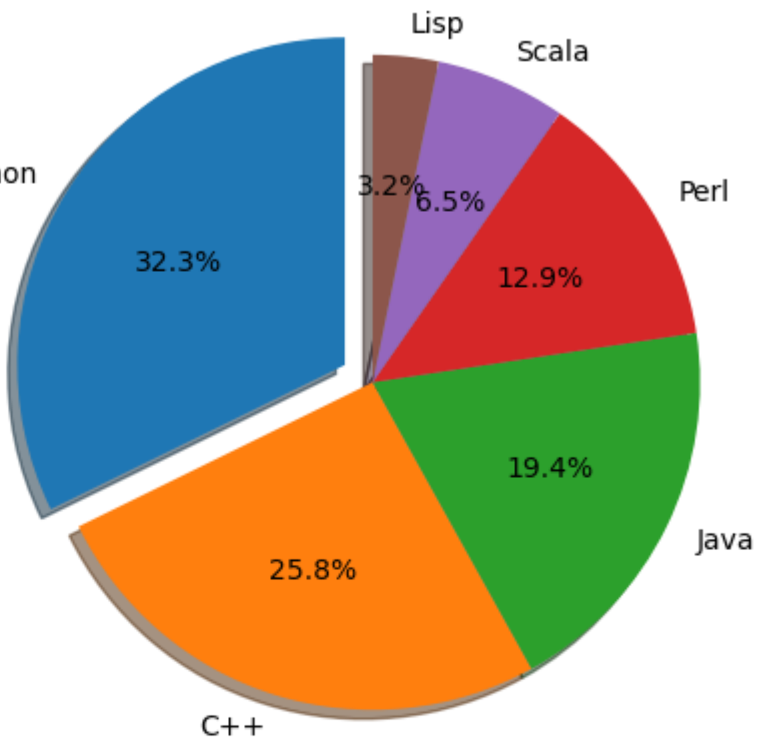
In [7]:

```
import matplotlib.pyplot as plt
import numpy as np

class Subject(object):
    def __init__(self):
        self._observers = []

    def attach(self, observer):
        if observer not in self._observers:
            self._observers.append(observer)

    def detach(self, observer):
        try:
            self._observers.remove(observer)
        except ValueError:
            pass

    def notify(self, modifier=None):
        for observer in self._observers:
            if modifier != observer:
                observer.update(self)

class Model(Subject):
    def __init__(self, name):
        Subject.__init__(self)
        self._name = name
        self._labels = ['Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp']
        self._data = [10,8,6,4,2,1]

    @property
    def labels(self):
        return self._labels

    @labels.setter
    def labels(self, labels):
        self._labels = labels
        self.notify()

    @property
    def data(self):
        return self._data

    @data.setter
    def data(self, data):
        self._data = data
        self.notify()

class View():
    def __init__(self, name=""):
        self._name = name

    def update(self, subject):
        print("Generalised Viewer '{}' has: \nName = {}; \nLabels = {}; \nData =

class TableView(View):
    def update(self, subject):
        fig = plt.figure(dpi=80)
        ax = fig.add_subplot(1,1,1)
        table_data = list(map(list,zip(subject._labels, subject._data)))
        table = ax.table(cellText=table_data, loc='center')
        table.set_fontsize(14)
        table.scale(1,4)
        ax.axis('off')
        ax.axis('off')
        plt.show()

class BarView(View):
    def update(self, subject):
        objects = subject._labels
        y_pos = np.arange(len(objects))
        performance = subject._data
        plt.bar(y_pos, performance, align='center', alpha=0.5)
        plt.xticks(y_pos, objects)
        plt.ylabel('Usage')
        plt.title('Programming language usage')
        plt.show()

# Pie chart viewer/ConcreteObserver - overrides the update() method
class PieView(View):
    def update(self, subject):
        labels = subject._labels
        sizes = subject._data
        explode = (0.1, 0, 0, 0, 0, 0)  # only "explode" the 1st slice
        fig1, ax1 = plt.subplots()
        ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', shadow
        ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a c
        plt.show()

# Create subjects
m1 = Model("Model 1")
m2 = Model("Model 2")

# Create observers
v1 = View("1: standard text viewer")
v2 = TableView("2: table viewer")
v3 = BarView("3: bar chart viewer")
v4 = PieView("4: pie chart viewer")  # New observer for pie chart

# Attach observers to the first model
m1.attach(v1)
m1.attach(v2)
m1.attach(v3)
m1.attach(v4)  # Attaching the new pie chart viewer

# Call the notify() method to see all the charts in their unchanged state
m1.notify()

# Change the properties of the first model
m1.labels = ['C#', 'PHP', 'JavaScript', 'ASP', 'Python', 'Smalltalk']
m1.data = [1, 18, 8, 60, 3, 1]
```

```
Generalised Viewer '1: standard text viewer' has:
Name = Model 1;
Labels = ['Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp'];
Data = [10, 8, 6, 4, 2, 1]
```

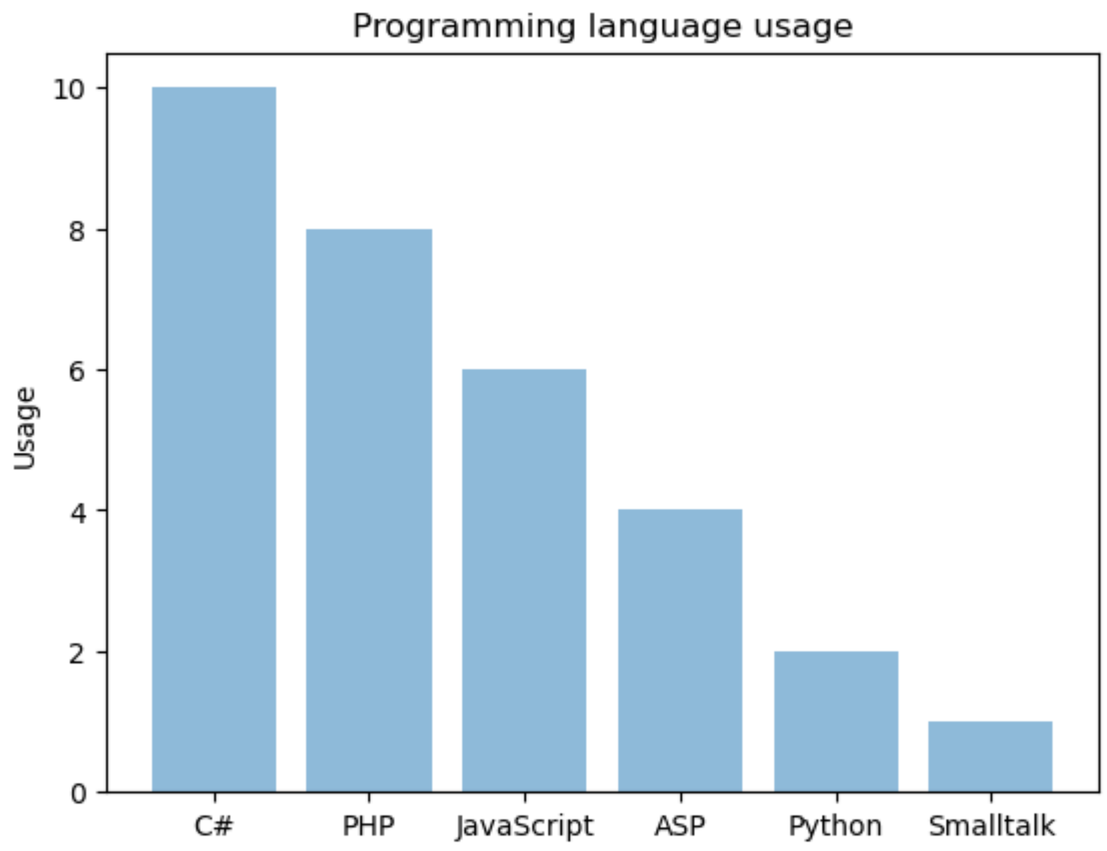
Python	10
C++	8
Java	6
Perl	4
Scala	2
Lisp	1

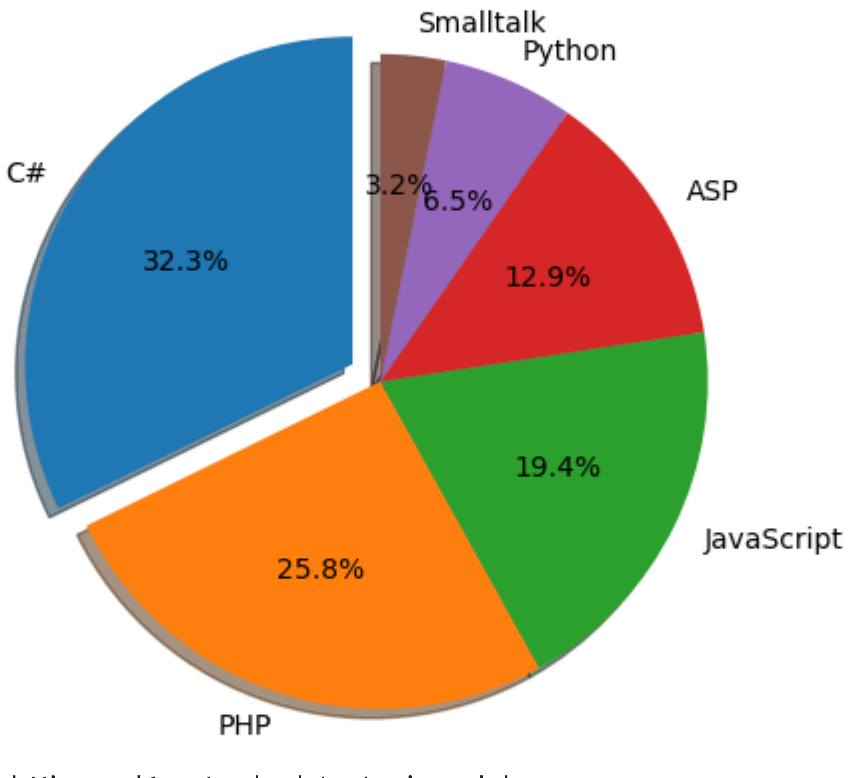




```
Generalised Viewer '1: standard text viewer' has:
Name = Model 1;
Labels = ['C#', 'PHP', 'JavaScript', 'ASP', 'Python', 'Smalltalk'];
Data = [10, 8, 6, 4, 2, 1]
```

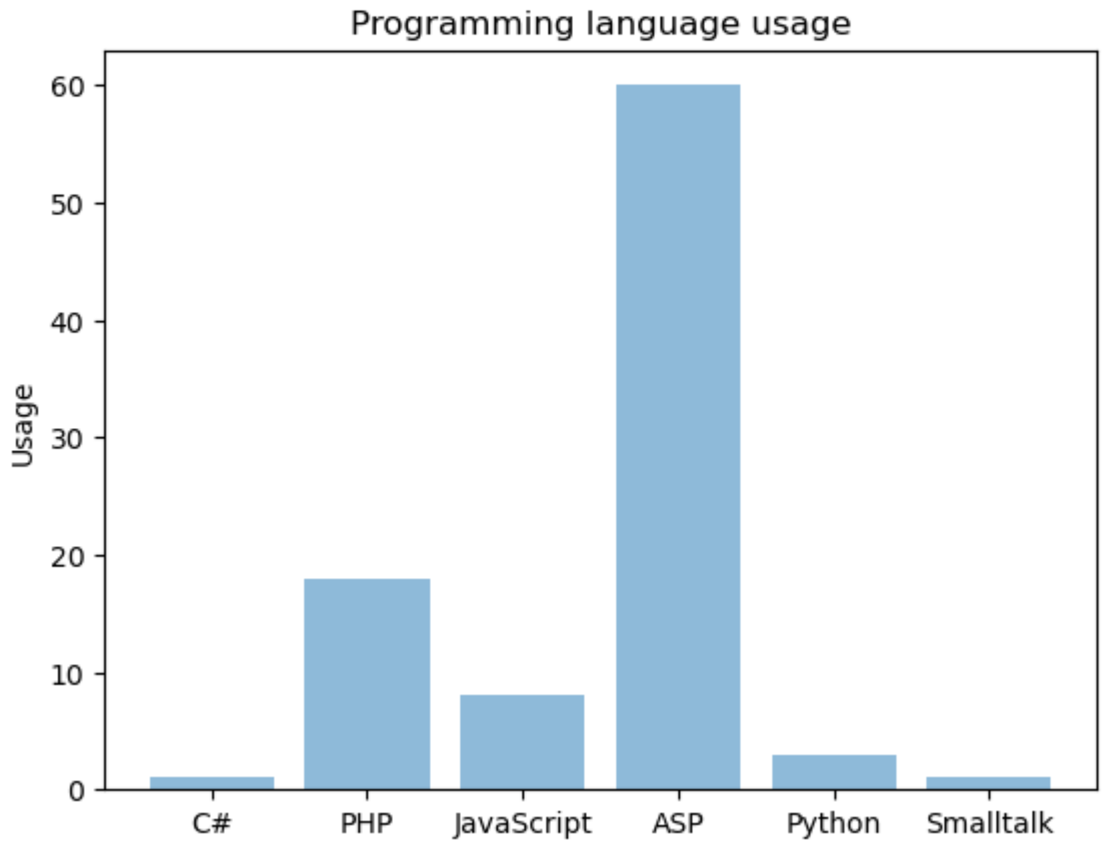
C#	10
PHP	8
JavaScript	6
ASP	4
Python	2
Smalltalk	1

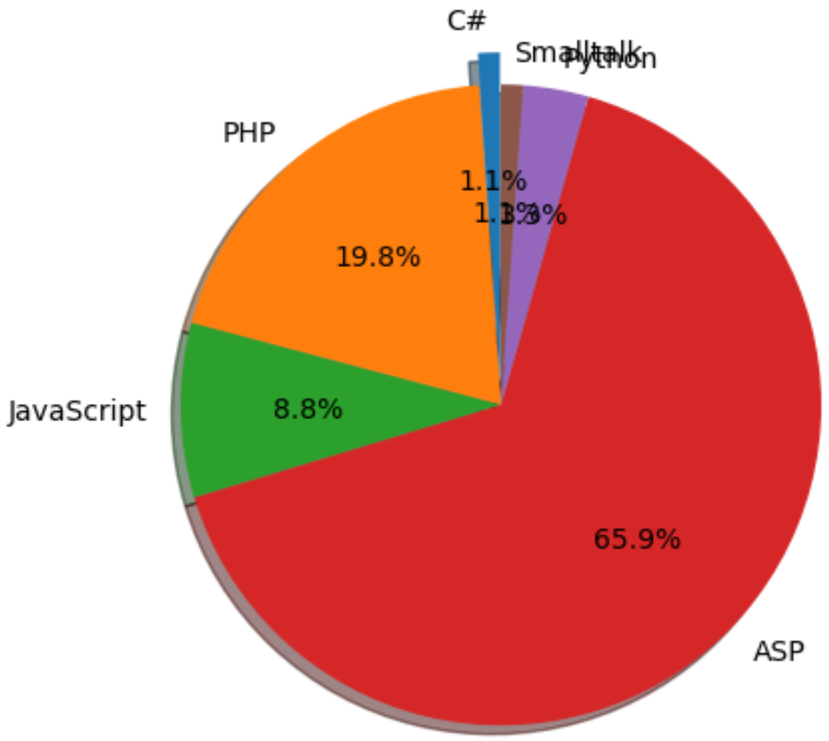




```
Generalised Viewer '1: standard text viewer' has:
Name = Model 1;
Labels = ['C#', 'PHP', 'JavaScript', 'ASP', 'Python', 'Smalltalk'];
Data = [1, 18, 8, 60, 3, 1]
```

C#	1
PHP	18
JavaScript	8
ASP	60
Python	3
Smalltalk	1




C#
Smalltalk
Python
PHP
1.1%
1.1%
3.3%
19.8%
JavaScript
8.8%
65.9%
ASP

Logbook Exercise 8

- Your task is to extend the modified version of Burkhard Meier's button factory (below) to create a text field factory
- In tkinter textfields are 'Entry' widgets
- Similary to the button factory structure you will need:
  - a concrete ***tk.Entry*** widget factory class - name it TextFactory()
  - the TextFactory's Factory Method - name it create_text(...)
  - an abstract product - name it TextBase() and give it default attributes 'textvariable' and 'background'
  - ... the constructor code for the tk.Entry widgets will be something like

```
    tk_string = tk.StringVar()
    tk_string.set(self.textvariable)
    super().__init__(frame, textvariable=tk_string,
background=self.background, foreground="white")
    self.grid(column=1, row=row)
```

  - 3x concrete text products
  - ... and assign them textvariable values of 'red type', 'blue type', "green type' respectively
  - ... and assign them background values of 'red', 'blue', and 'green' respectively
- To extend the OOP class with a create_text_fields() method
  - ... that creates a factory object
  - ... and the factory method will need to be called to create each text widget
- the end product should look something like this ...

Challenge question

- At present the example has a 2x 'concrete' creators. As a comment just above the line `import tkinter as tk` briefly explain what you would need to do to refactor code so that the concrete creators extended an abstract class/interface called ***Creator*** (i.e. just as it appears in Gamma et al. (1995) and on slide 2)

In [8]:
```python
# To refactor the code I'd introduce an abstract 'Creator' class with a factory
# method 'FactoryMethod()'. 'ButtonFactory' and 'TextFactory' would inherit from
# and implement this method to instantiate 'ButtonBase' and 'TextBase' products.
# allows for encapsulating object creation and adhering to the pattern shown in

import tkinter as tk
from tkinter import ttk

# Button Base remains unchanged
class ButtonBase(tk.Button):
    title      = 'Button'
    relief     = 'flat'
    foreground = 'white'

    def __init__(self, frame, row):
        super().__init__(frame, text=self.title, relief=self.relief, foreground=
        self.grid(column=0, row=row)

# Concrete Product buttons remain unchanged
class ButtonRidge(ButtonBase):
    title      = 'Ridge Button'
    relief     = 'ridge'
    foreground = 'red'

class ButtonSunken(ButtonBase):
    title      = 'Sunken Button'
    relief     = 'sunken'
    foreground = 'blue'

class ButtonGroove(ButtonBase):
    title      = 'Groove Button'
    relief     = 'groove'
    foreground = 'green'

# ButtonFactory remains unchanged
class ButtonFactory():
    _button_classes = [ButtonRidge, ButtonSunken, ButtonGroove]

    def create_button(self, frame, index):
        button_class = self._button_classes[index]
        return button_class(frame, index)

# Abstract Product for Text Fields
class TextBase(tk.Entry):
    textvariable = 'Default Text'
    background     = 'white'

    def __init__(self, frame, row):
        tk_string = tk.StringVar()
        tk_string.set(self.textvariable)
        super().__init__(frame, textvariable=tk_string, background=self.backgrou
        self.grid(column=1, row=row)

# Concrete text products
class TextRedType(TextBase):
    textvariable = 'red type'
    background     = 'red'

class TextBlueType(TextBase):
    textvariable = 'blue type'
    background     = 'blue'

class TextGreenType(TextBase):
```

```
class TextGreenType(TextBase):
    textvariable = 'green type'
    background     = 'green'

# Concrete Text Factory
class TextFactory():
    _text_classes = [TextRedType, TextBlueType, TextGreenType]

    def create_text(self, frame, index):
        text_class = self._text_classes[index]
        return text_class(frame, index)

# Extended OOP class with create_text_fields method
class OOP():
    def __init__(self):
        self.window = tk.Tk()
        self.window.title("Python GUI")
        self.create_widgets()

    def create_widgets(self):
        self.frame = ttk.LabelFrame(text=' Widget Factory ')
        self.frame.grid(padx=40, pady=10)

        self.create_buttons()
        self.create_text_fields()

    def create_buttons(self):
        factory = ButtonFactory()
        for i in range(3):
            factory.create_button(self.frame, i)

    def create_text_fields(self):
        factory = TextFactory()
        for i in range(3):
            factory.create_text(self.frame, i)

oop = OOP()
oop.window.mainloop()
```

Logbook Exercise 9

- Extend the Jungwoo Ryoo's Abstract Factory below to mirror the structure used by a statically typed languages by:
  - adding a 'CatFactory' and a 'Cat' class with methods that are compatible with 'DogFactory' and 'Dog' respectively
  - providing an Abstract Factory class/interface named 'AnimalFactory' and make both the Dog and Cat factories implement this
  - providing an AbstractProduct (name this 'Animal') and make both Dog and cat classes implement this
- Use in-code comments (#) to identify the abstract and concrete entities present in Gamma et al. (1995)
  - comments should include: "# Abstract Factory #"; "# Concrete Factory #"; "# Abstract Product #"; "# Concrete Product #"; and "# The Client #"
- Implement the CatFactory ... the end output should look something like this ...

```
Our pet is 'Dog'!
Our pet says hello by 'Woof'!
Our pet says hello by 'Woof'!
```

In [9]:
```python
# Abstract Factory #
class AnimalFactory:
    def get_pet(self):
        pass
    def get_food(self):
        pass

# Abstract Product #
class Animal:
    def speak(self):
        pass
    def __str__(self):
        pass

# Concrete Factory #
class DogFactory(AnimalFactory):
    def get_pet(self):
        """Returns a Dog object"""
        return Dog()
    def get_food(self):
        """Returns a Dog Food object"""
        return "Dog Food!"

# Concrete Factory #
class CatFactory(AnimalFactory):
    def get_pet(self):
        """Returns a Cat object"""
        return Cat()
    def get_food(self):
        """Returns a Cat Food object"""
        return "Cat Food!"

# Concrete Product #
class Dog(Animal):
    def speak(self):
        return "Woof!"
    def __str__(self):
        return "Dog"

# Concrete Product #
class Cat(Animal):
    def speak(self):
        return "Meeoowww"
    def __str__(self):
        return "Cat"

# The Client #
class PetStore:
    """ PetStore houses our Abstract Factory """
    def __init__(self, pet_factory=None):
        """ pet_factory is our Abstract Factory """
        self._pet_factory = pet_factory
    def show_pet(self):
        """ Utility method displays details of objects returned by Factory """
        pet = self._pet_factory.get_pet()
        pet_food = self._pet_factory.get_food()
        print("Our pet is '{}'!".format(pet))
        print("Our pet says hello by '{}'".format(pet.speak()))
        print("Its food is '{}'!".format(pet_food))

# Working with the DogFactory
dog_factory = DogFactory()
dog_shop = PetStore(dog_factory)
dog_shop = PetStore(dog_factory)
dog_shop.show_pet()

print()

# Working with the CatFactory
cat_factory = CatFactory()
cat_shop = PetStore(cat_factory)
cat_shop.show_pet()
```

```
Our pet is 'Dog'!
Our pet says hello by 'Woof!'
Its food is 'Dog Food!'!

Our pet is 'Cat'!
Our pet says hello by 'Meeoowww'
Its food is 'Cat Food!'!
```

Logbook Exercise 10

- Modify Jungwoo Ryoo's Strategy Pattern to showcase ***OpenCV*** capabilities with different image processing strategies
- We will use the ***OpenCV*** (Open Computer Vision) library which has been reproduced with Python bindings
- OpenCV has many standard computer science image-processing filters and includes powerful AI machine learning algorithms
- The following resources provide more information on OpenCv with Python ...
  - Beyeler, M. (2015). OpenCV with Python blueprints. Packt Publishing Ltd.
  - Joshi, P. (2015). OpenCV with Python by example. Packt Publishing Ltd.
  - The cartoon effect is from http://www.askaswiss.com/2016/01/how-to-create-cartoon-effect-opencv-python.html and https://www.tutorialspoint.com/cartooning-an-image-using-opencv-in-python

Development stages

- Install the OpenCV package - we have to do this manually ...
  - Start Anaconda Navigator
  - From Anaconda run the CMD.exe terminal
  - At the prompt type ... `conda install opencv-python`
  - The process may pause with a prompt ... `Proceed ([y]/n)?` ... just accept this ... `y`
- Copy Jungwoo Ryoo's code to a code cell below this one
- As well as the ***types*** module you will need to provide access to OpenCV and numpy as follows
```
# Import OpenCV
import cv2
import numpy as np
```
- Please place a copy of clouds.jpg in the same directory as your Jupyter logbook
- The code for each strategy and some notes on implementing these are below ...
- The output should look something like this ... but if you wish feel free to experiment with something else ... cats etc.!

Implementing image processing strategies

- There will be six strategy objects s0-s5, where s0 is the default strategy of the **Strategy** class
- Instead of assigning a name to each strategy object, you will need to reference the image to be processed - 'clouds.jpg'
  - i.e. s0.image = "clouds.jpg"
- The *body* code for each strategy is below, you will need to provide the method signatures and their executions

strategy s0

The default ***execute()*** method that simply displays the image sent to it

```
            print("The image {} is used to execute Strategy 0 -
    Display image".format(self.image))
            img_rgb = cv2.imread(self.image)
            cv2.imshow('Image', img_rgb)
```

strategy s1

This converts a colour image into a mononochrome one - suggested strategy method name is ***strategy_greyscale***

```
        img_rgb = cv2.imread(self.image)
        img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
        cv2.imshow('Greyscale image', img_gray)
```

strategy s2

This applies a blur filter to an image - suggested strategy method name is ***strategy_blur***

```
        img_rgb = cv2.imread(self.image)
        img_blur = cv2.medianBlur(img_rgb, 7)
        cv2.imshow('Blurred image', img_blur)
```

strategy s3

This produces a colour negative image from a colour one - suggested strategy method name is ***strategy_colNegative***

```
      img_rgb = cv2.imread(self.image)
        for x in np.nditer(img_rgb, op_flags=['readwrite']):
            x[...] = (255 - x)
        cv2.imshow('Colour negative', img_rgb)
```

strategy s4

This produces a monochrome negative image from a colour one - suggested strategy method name is ***strategy_greyNegative***

```
        img_rgb = cv2.imread(self.image)
        img_grey = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
        for x in np.nditer(img_grey, op_flags=['readwrite']):
            x[...] = (255 - x)
        cv2.imshow('Monochrome negative', img_grey)
```

strategy s5

This produces a cartoon-like effect - suggested strategy method name is ***strategy_cartoon***

```
        #Use bilateral filter for edge smoothing.
        num_down = 2 # number of downsampling steps
        num_bilateral = 7 # number of bilateral filtering steps
        img_rgb = cv2.imread(self.image)
        # downsample image using Gaussian pyramid
        img_color = img_rgb
        for _ in range(num_down):
            img_color = cv2.pyrDown(img_color)
        # repeatedly apply small bilateral filter instead of
    applying one large filter
        for _ in range(num_bilateral):
            img_color = cv2.bilateralFilter(img_color, d=9,
    sigmaColor=9, sigmaSpace=7)
        # upsample image to original size
        for _ in range(num_down):
            img_color = cv2.pyrUp(img_color)
        #Use median filter to reduce noise convert to grayscale and
    apply median blur
        img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
        img_blur = cv2.medianBlur(img_gray, 7)
        #Use adaptive thresholding to create an edge mask detect and
    enhance edges
        img_edge = cv2.adaptiveThreshold(img_blur, 255,
    cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, blockSize=9, C=2)
        # Combine color image with edge mask & display picture,
    convert back to color, bit-AND with color image
        img_edge = cv2.cvtColor(img_edge, cv2.COLOR_GRAY2RGB)
        img_cartoon = cv2.bitwise_and(img_color, img_edge)
        # display
        cv2.imshow("Cartoon-ised image", img_cartoon);
    cv2.waitKey(0); cv2.destroyAllWindows()
```

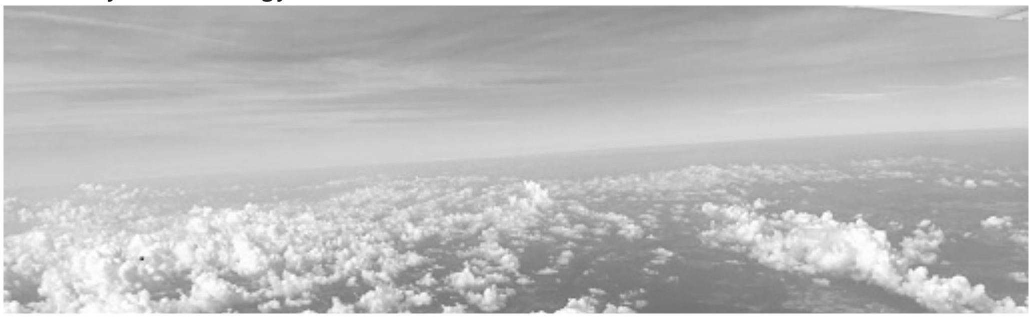
In [10]:
```
import types
import cv2
import numpy as np

class Strategy:
    def __init__(self, function=None):
        self.name = "Default Strategy"
        self.image = "clouds.jpg"

        if function:
            self.execute = types.MethodType(function, self)

    def execute(self):
        print("The image {} is used to execute Strategy 0 - Display image".forma
        img_rgb = cv2.imread(self.image)
        # Saving the original image as it's shown
        cv2.imwrite(self.name.replace(" ", "_") + ".jpg", img_rgb)
        cv2.imshow('Image', img_rgb)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

# Updated strategy methods to include saving the processed images
def strategy_greyscale(self):
    print("Executing strategy for Greyscale conversion")
    img_rgb = cv2.imread(self.image)
    img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
    cv2.imwrite(self.name.replace(" ", "_") + ".jpg", img_gray)  # Save the imag
    cv2.imshow('Greyscale image', img_gray)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def strategy_blur(self):
```

```
def strategy_blur(self):
    print("Executing strategy for Blurring")
    img_rgb = cv2.imread(self.image)
    img_blur = cv2.medianBlur(img_rgb, 7)
    cv2.imwrite(self.name.replace(" ", "_") + ".jpg", img_blur)  # Save the imag
    cv2.imshow('Blurred image', img_blur)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def strategy_colNegative(self):
    print("Executing strategy for Color Negative")
    img_rgb = cv2.imread(self.image)
    img_neg = 255 - img_rgb
    cv2.imwrite(self.name.replace(" ", "_") + ".jpg", img_neg)  # Save the image
    cv2.imshow('Colour negative', img_neg)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def strategy_greyNegative(self):
    print("Executing strategy for Grey Negative")
    img_rgb = cv2.imread(self.image)
    img_grey = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
    img_neg = 255 - img_grey
    cv2.imwrite(self.name.replace(" ", "_") + ".jpg", img_neg)  # Save the image
    cv2.imshow('Monochrome negative', img_neg)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def strategy_cartoon(self):
    print("Executing strategy for Cartoon effect")
    num_down = 2
    num_bilateral = 7
    img_rgb = cv2.imread(self.image)
    img_color = img_rgb
    for _ in range(num_down):
        img_color = cv2.pyrDown(img_color)
    for _ in range(num_bilateral):
        img_color = cv2.bilateralFilter(img_color, d=9, sigmaColor=9, sigmaSpace
    for _ in range(num_down):
        img_color = cv2.pyrUp(img_color)
    img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
    img_blur = cv2.medianBlur(img_gray, 7)
    img_edge = cv2.adaptiveThreshold(img_blur, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
    img_edge = cv2.cvtColor(img_edge, cv2.COLOR_GRAY2RGB)
    img_cartoon = cv2.bitwise_and(img_color, img_edge)
    cv2.imwrite(self.name.replace(" ", "_") + ".jpg", img_cartoon)  # Save the i
    cv2.imshow("Cartoon-ised image", img_cartoon)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Application of strategies as before
s0 = Strategy()
s1 = Strategy(strategy_greyscale)
s1.name = "Greyscale Strategy"
s2 = Strategy(strategy_blur)
s2.name = "Blur Strategy"
s3 = Strategy(strategy_colNegative)
s3.name = "Color Negative Strategy"
s4 = Strategy(strategy_greyNegative)
s4.name = "Grey Negative Strategy"
s5 = Strategy(strategy_cartoon)
s5.name = "Cartoon Strategy"

strategies = [s0, s1, s2, s3, s4, s5]
```

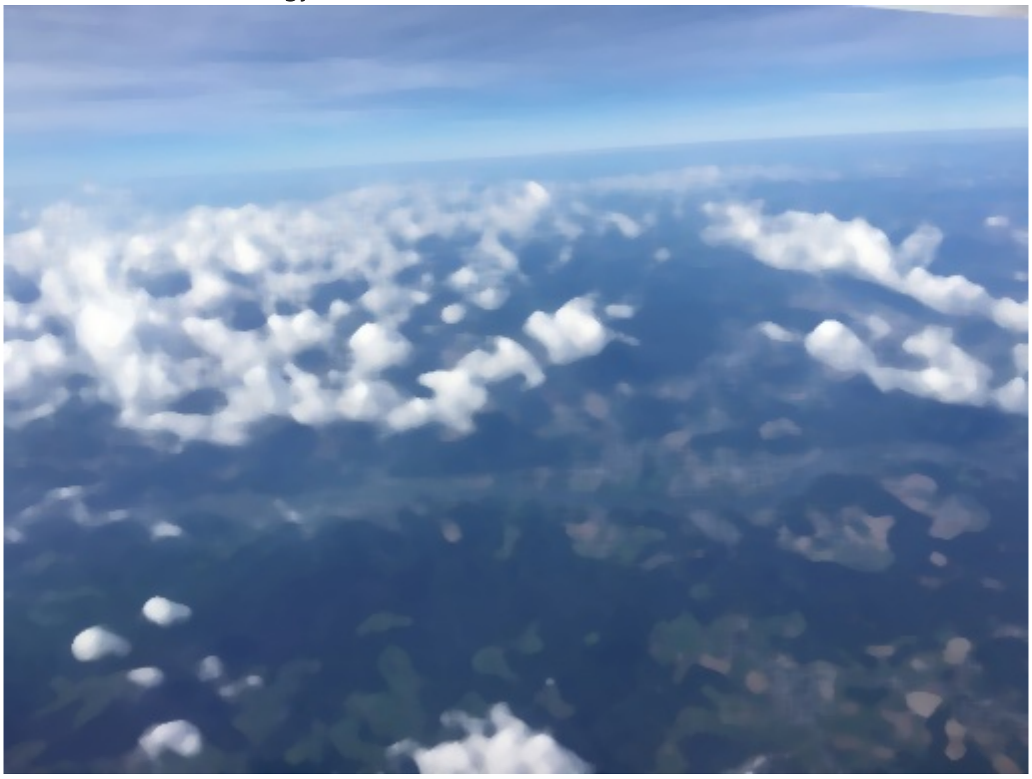
```
for strategy in strategies:
    strategy.execute()
```
Clouds.jpg

Clouds with default Strategy

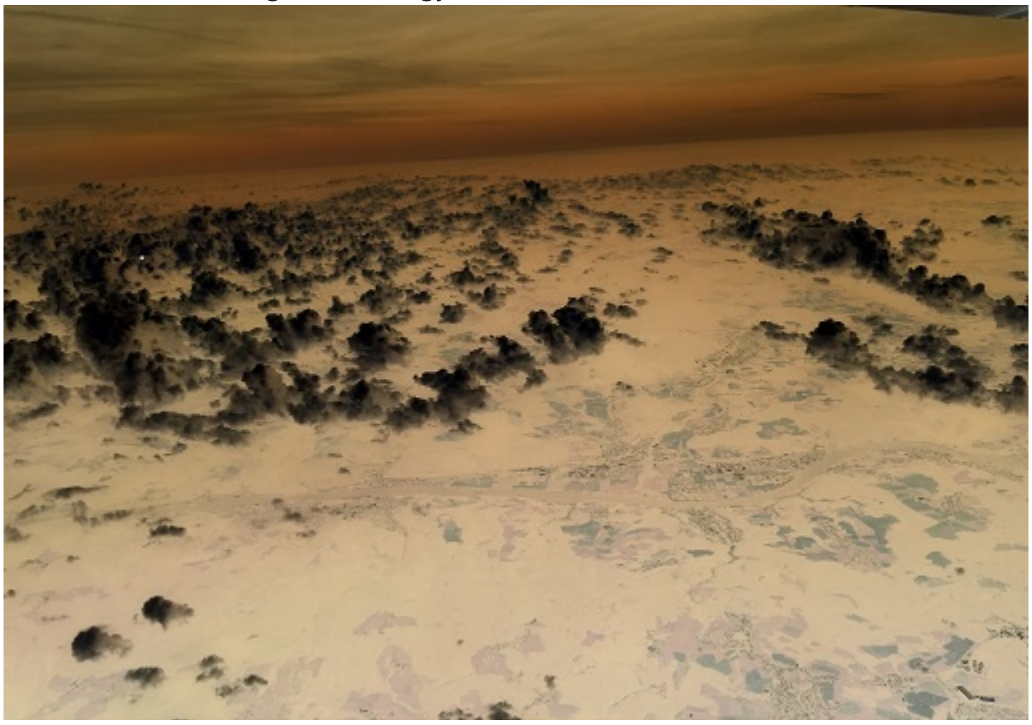
Cloud with Greyscale Strategy

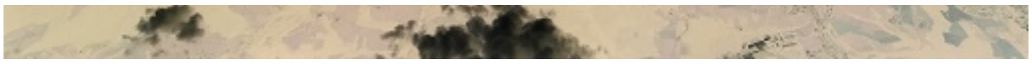


Clouds with Blur Strategy

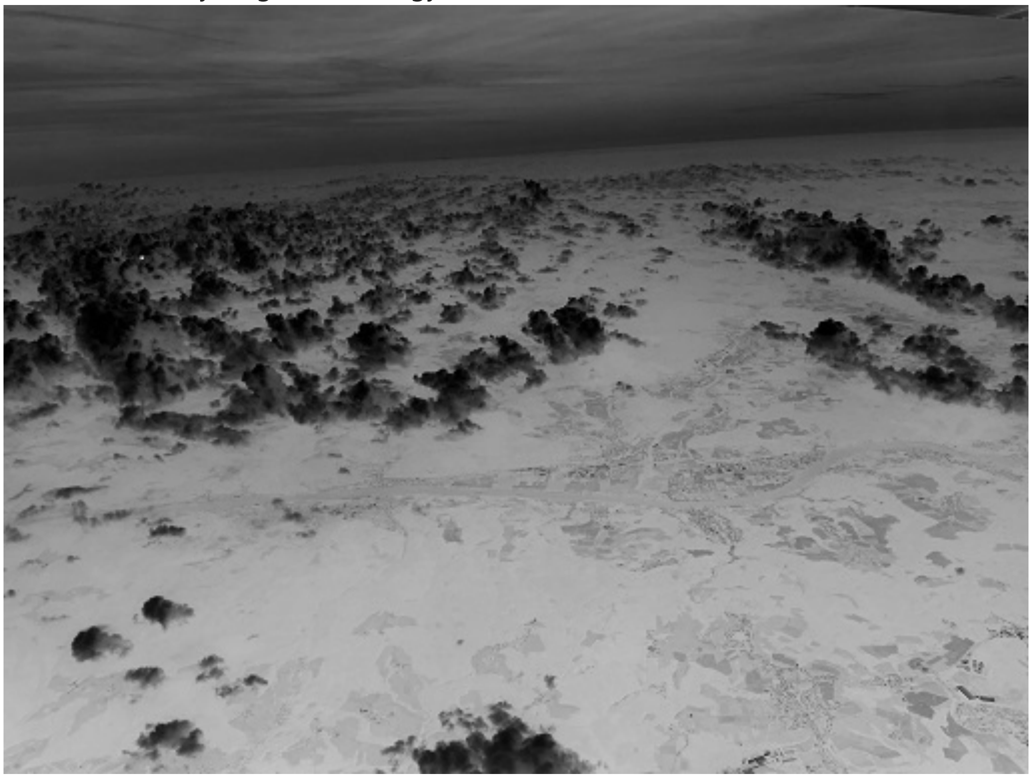


Clouds with Color Negative Strategy

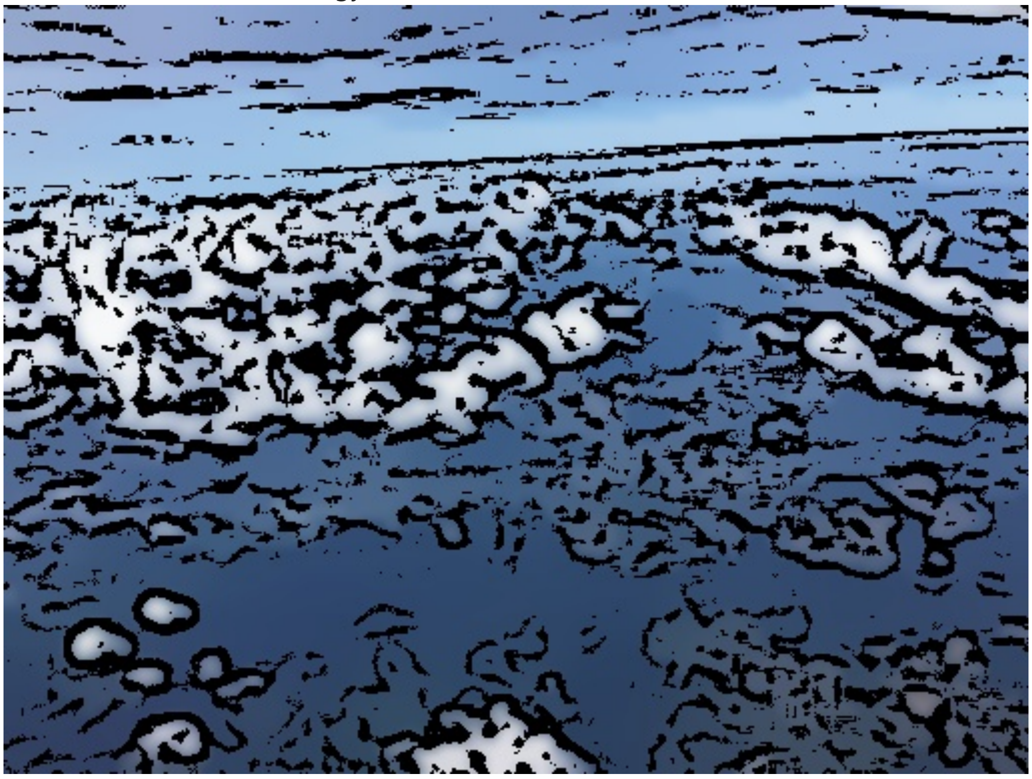




Clouds with Grey Negative Strategy



Clouds with Cartoon Strategy



Logbook Exercise 11

- Demonstrate the use of `__iter__()`, `__next()__` and `StopIteration` using ...
- ... the first four items from the top10books list (see above) ...
- ... and the following structure

```
list = [lit 1, lit 2, lit 3]
```

```
    mylist = ['item1', 'item2', 'item3']
```

In [11]:
```python
import sys


class BookIterator:
    def __init__(self, books):
        self.books = books
        self.index = 0

    def __iter__(self):
        # The __iter__() method returns the iterator object itself.
        return self

    def __next__(self):
        # The __next__() method should return the next item in the sequence.
        if self.index < len(self.books):
            result = self.books[self.index]
            self.index += 1
            return result
        else:
            # Once all items have been returned, __next__() should raise StopIte
            raise StopIteration


top3books = ["Anna Karenina by Leo Tolstoy", "Madame Bovary by Gustave Flaubert"

book_iterator = BookIterator(top3books)

# Using the iterator
try:
    print(next(book_iterator))
    print(next(book_iterator))
    print(next(book_iterator))
    print(next(book_iterator))
    # The next call should exceed the number of items and raise a StopIteration
    print(next(book_iterator))
except StopIteration as e:
    print(f"All books have been iterated, StopIteration has been raised: {sys.ex
```

```
Anna Karenina by Leo Tolstoy
Madame Bovary by Gustave Flaubert
War and Peace by Leo Tolstoy
All books have been iterated, StopIteration has been raised: (<class 'StopIterat
ion'>, StopIteration(), <traceback object at 0x00000217DB17A900>)
```

Logbook Exercise 12 - The Adapter DP

- Modify Jungwoo Ryoo's Adapter Pattern example (the one with 'country' classes that 'speak' greetings) to showcase:
  - the ***polymorphic*** capability of the Adpater DP
  - the geo-data capabilities of ***matplotlib geographical projections*** ...
  - in combination with ***Cartopy geospatial data processing*** package to ***produce maps and other geospatial data analyses***.
- A frequent problem in handling geospatial data is that the user often needs to convert it from one form of map projection (essentially a formula to convert the globe into a plane for map-representation) to another map projection

- Fortunately other clever people have written the algorithms we need
- Less fortunately, the interfaces of all the classes that return projections are different

Development Stages

- We need ...

  - first, to install the Python cartographic ***Cartopy*** package. In Anaconda launch a CMD.exe terminal and enter the following ...
    `conda install -c conda-forge cartopy`
  - to insert a code cell below this one ... and copy the extended example of Ryoo's Adapter above (with 'speak' methods in Korean, British and German) in this ...
  - an ***Adapter*** - Ryoo's adapter is already a well-engineered solution that requires no modification
  - then to import some essential packages
    `import cartopy.crs as ccrs`
    `import matplotlib.pyplot as plt`
  - then add the 'adaptee' classes - here represented by the plot axes and their map projections

```
class PlateCarree:
    def __init__(self):
        self.name = "PlateCarree"
    def project_PlateCarree(self):
        ax = plt.axes(projection=ccrs.PlateCarree())
        return ax

class InterruptedGoodeHomolosine:
    def __init__(self):
        self.name = "InterruptedGoodeHomolosine"
    def project_InterruptedGoodeHomolosine(self):
        ax =
plt.axes(projection=ccrs.InterruptedGoodeHomolosine())
        return ax

class AlbersEqualArea:
    def __init__(self):
        self.name = "AlbersEqualArea"
    def project_AlbersEqualArea(self):
        ax = plt.axes(projection=ccrs.AlbersEqualArea())
        return ax

class Mollweide:
    def __init__(self):
        self.name = "Mollweide"
    def project_Mollweide(self):
        ax = plt.axes(projection=ccrs.Mollweide())
        return ax
```

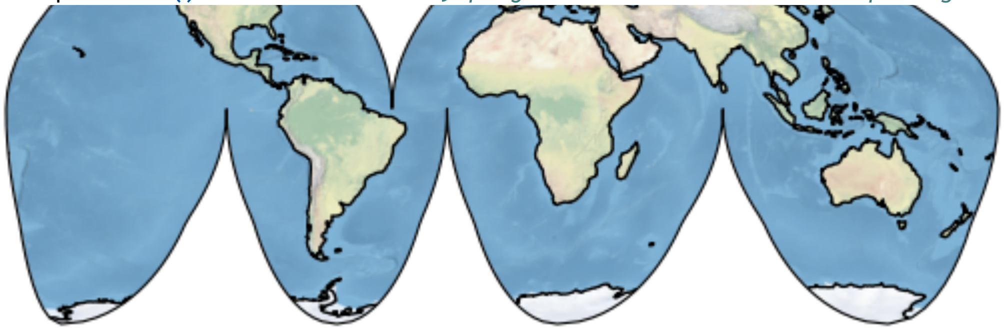
- similarly to Ryoo's example you will need a collection to store projection objects
- again, simlarly to Ryoo, to create all the projection objects (e.g. `plateCarree =PlateCarree()`)
- again, similarly to Ryoo, to append to the collection key-value pairs for each projection and its projection method
- finally to traverse the list of objects to:

```
  # Create an axes with the specified projection
  ax = obj.project()
  # Attach Cartopy's default geospatially registered map/image of
the world
  ax.stock_img()
  # Add the coastlines - highlight these with a black vector
  ax.coastlines()
  # Print the name of the object/projection
  print(obj.name)
  # Plot the axes, projection and render the map-image to the
projection
  plt.show()
```

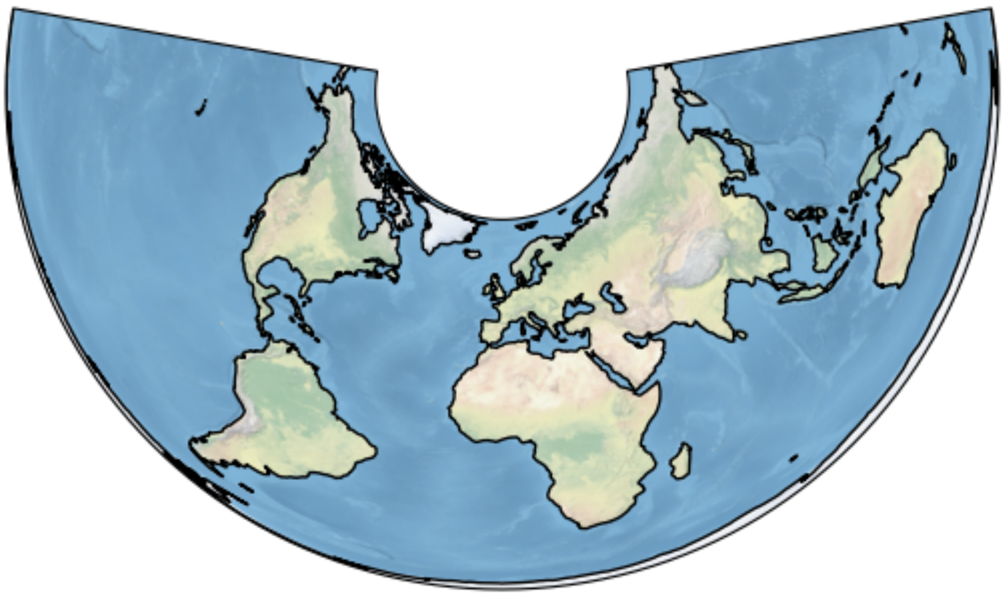
- The output should look something like this ...

In [12]:
```python
# Essential Imports
import cartopy.crs as ccrs
import matplotlib.pyplot as plt

# Adaptee Classes for different projections
class PlateCarree:
    def __init__(self):
        self.name = "PlateCarree"
    def project(self):
        ax = plt.axes(projection=ccrs.PlateCarree())
        return ax

class InterruptedGoodeHomolosine:
    def __init__(self):
        self.name = "InterruptedGoodeHomolosine"
    def project(self):
        ax = plt.axes(projection=ccrs.InterruptedGoodeHomolosine())
        return ax

class AlbersEqualArea:
    def __init__(self):
        self.name = "AlbersEqualArea"
    def project(self):
        ax = plt.axes(projection=ccrs.AlbersEqualArea())
        return ax
```

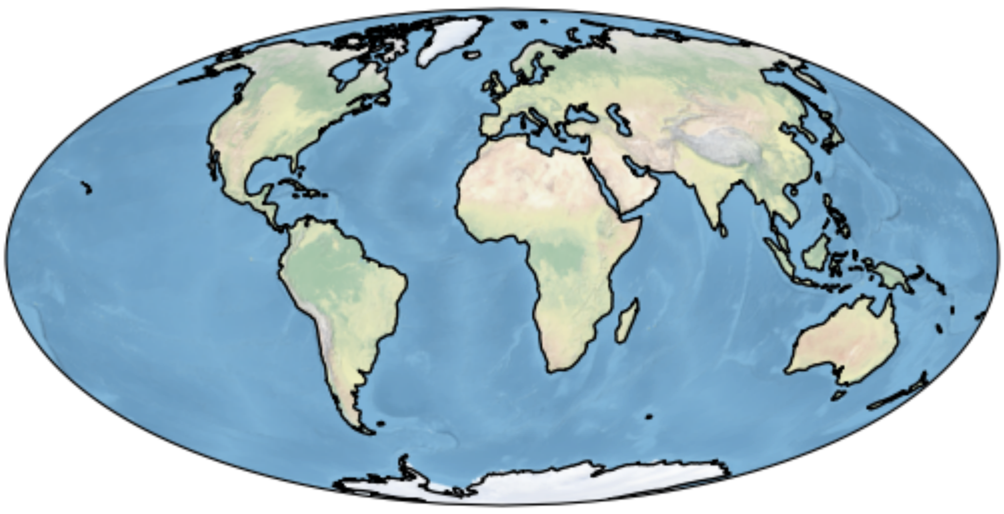
```
class Mollweide:
    def __init__(self):
        self.name = "Mollweide"
    def project(self):
        ax = plt.axes(projection=ccrs.Mollweide())
        return ax

# Create projection objects
plateCarree = PlateCarree()
interruptedGoodeHomolosine = InterruptedGoodeHomolosine()
albersEqualArea = AlbersEqualArea()
mollweide = Mollweide()

# Collection to store projection objects and their methods
projections = [plateCarree, interruptedGoodeHomolosine, albersEqualArea, mollwei

# Iterate over the collection to render each map projection
for obj in projections:
    ax = obj.project()  # Create an axes with the specified projection
    ax.stock_img()  # Attach Cartopy's default geospatially registered map/image
    ax.coastlines()  # Add the coastlines - highlight these with a black vector
    print(obj.name)  # Print the name of the object/projection
    plt.show()  # Plot the axes, projection and render the map-image to the proj
```



```
AlbersEqualArea
```



```
Mollweide
```



Logbook Exercise 13 - The Decorator DP

- Repair the code below so that the decorator reveals the name and docstring of aTestMethod()
- Note ... the @wrap decorator is NOT needed here

```
    <<< Name of the 'decorated' function ...  aTestMethod  >>>
    <<< Docstring for the 'decorated' function is ...  This is a
method to test the docStringDecorator  >>>
    What is your name? ... Buggy Code
    Hello ... Buggy Code
```

In [13]:
```
def docStringDecorator(f):
    '''Decorator that automatically reports name and docstring for a decorated f
    def wrapped_function(*args, **kwargs):
        print("<<< Name of the 'decorated' function ... ", f.__name__, " >>>")
        print("<<< Docstring for the 'decorated' function is ... ", f.__doc__, "
        return f(*args, **kwargs)
    return wrapped_function

@docStringDecorator
def aTestMethod():
    '''This is a method to test the docStringDecorator'''
    nm = input("What is your name? ... ")
    msg = "Hello ... " + nm
    return msg

print(aTestMethod())
```

```
<<< Name of the 'decorated' function ...  aTestMethod  >>>
<<< Docstring for the 'decorated' function is ...  This is a method to test the
docStringDecorator  >>>
What is your name? ... Dominic
Hello ... Dominic
```

Logbook Exercise 14 - The 'conventional' Singleton DP

- Insert a code cell below here
- Copy the code from Dusty Philips' singleton
- Create two objects
- Test output using `print( ... )` and `repr( ... )` as well as the `==` operator to determine whether or not the two objects are the same and occupy the same memory addresses
- Make a note below of your findings

In [14]:
```python
## SINGLETON - Extended from Dusty Philips (2015) ##
class OneOnly:
    _singleton = None
    def __new__(cls, *args, **kwargs):
        if not cls._singleton:
            cls._singleton = super(OneOnly, cls).__new__(cls, *args, **kwargs)
        return cls._singleton

p1 = OneOnly()
p2 = OneOnly()

# Testing the output and comparison
print(p1)  # Print the first object
print(p2)  # Print the second object
print(repr(p1))  # Detailed representation of the first object
print(repr(p2))  # Detailed representation of the second object
print(p1 == p2)  # Check if both objects are the same
print(id(p1) == id(p2))  # Check if both objects occupy the same memory address
```

```
<__main__.OneOnly object at 0x00000217D61D9460>
<__main__.OneOnly object at 0x00000217D61D9460>
<__main__.OneOnly object at 0x00000217D61D9460>
<__main__.OneOnly object at 0x00000217D61D9460>
True
True
```

My observations having tested the two objects are ...

The output of both print and repr functions for the two objects are the same, showing that both variables refer to the same instance of the OneOnly class. The memory address 0x00000217D61D9460 is shown for both p1 and p2, demonstrating that there is only one instance of the OneOnly class being referred to by both variables.

The equality check (p1 == p2) returns True, again showing that p1 and p2 are not just instances of the same class but refer to the same instance of the class. Additionally, the memory address comparison (id(p1) == id(p2)) also returns True, providing further proof that both variables point to the same object in memory. This behaviour is a direct consequence of the Singleton design pattern, which restricts the instantiation of a class to one "single" instance and provides a global point of access to that instance.

The Singleton pattern implementation via the **new** method in Python effectively ensures that subsequent attempts to create new instances of the OneOnly class will always return the same instance. This pattern can be beneficial when one object is needed to coordinate actions across the system, such as configuration management, logging, or accessing a shared resource.

The output demonstrates that the Singleton pattern has been implemented correctly. It ensures that only a single instance of the OneOnly class is created and shared, regardless of the number of times an attempt is made to instantiate the class.

Logbook Exercise 15 - The 'Borg' Singleton DP

- Repeat the exercise above ...
- Insert a code cell below here
- Copy the code from Alex Martelli's 'Borg' singleton
- Create THREE objects ... ***NOTE***: pass a name for the object when you call the constructor
- Test output using `print( ... )` and `repr( ... )` as well as the `==` operator to determine whether or not the objects are the same and occupy the same memory addresses
- ***Also*** can you use `print( ... )` to test the assertion in the notes above that ... "- `_shared_state` is effectively static and is only created once, when the first singleton is instantiated "
- Make a note below of your findings

My observations having tested the three objects are ...

The Singleton design pattern ensures that a class has only one instance while providing a global method to access this instance. As implemented in the code below, the' Borg' Singleton variant offers a twist on this concept: instead of enforcing a single instance of a class, it ensures that all instances share the same state. The observed output from the Borg pattern implementation highlights three key aspects:

- Shared State: Although Singleton objects s1, s2, and s3 are distinct instances (as confirmed by their unique memory addresses), they share the same state. This is evident from the update of _shared_state with each new instance and the consistent output across different objects.

- Individual Instances: Despite sharing the same state, the objects are not identical. They are separate instances, each with its own unique memory address, demonstrating the 'Borg' Singleton pattern's approach to the Singleton concept.

- Global State Accessibility: The last state set by any instance updates the shared state for all instances, showing how the Borg pattern facilitates global state accessibility across multiple object instances.

In summary, the 'Borg' Singleton pattern maintains the principle of controlled access to a shared resource (the state) across multiple instances, changing from the traditional Singleton's single instance constraint. This approach is beneficial in scenarios where instance uniqueness is secondary to the need for shared state across instances.

In [15]:

```python
# Singleton/BorgSingleton.py
# Alex Martelli's 'Borg'

class Borg:
    _shared_state = {}

    def __init__(self):
        self.__dict__ = self._shared_state
        print("Value of self._shared_state is ..."+str(self._shared_state))

class Singleton(Borg):
    def __init__(self, arg):
        # Here the 'static' Borg class is updated with the state of the new sing
        Borg.__init__(self)
        self.val = arg
    def __str__(self):
        return self.val

# Creating three Singleton objects
s1 = Singleton("s1")
s2 = Singleton("s2")
s3 = Singleton("s3")

# Testing the output and comparison
print(s1)  # Print the first object's value
print(s2)  # Print the second object's value
print(s3)  # Print the third object's value
print(repr(s1))  # Detailed representation of the first object
print(repr(s2))  # Detailed representation of the second object
print(repr(s3))  # Detailed representation of the third object
print(s1 == s2, s2 == s3)  # Check if objects are the same
print(id(s1), id(s2), id(s3))  # Check memory addresses

# Additional check to validate shared state assertion
print(s1._shared_state)
```

```
Value of self._shared_state is ...{}
Value of self._shared_state is ...{'val': 's1'}
Value of self._shared_state is ...{'val': 's2'}
s3
s3
s3
<__main__.Singleton object at 0x00000217D61D83A0>
<__main__.Singleton object at 0x00000217D6176A30>
<__main__.Singleton object at 0x00000217DAF223D0>
False False
2301399761824 2301399362096 2301480805328
{'val': 's3'}
```