Academic Integrity Statement

As a matter of Departmental policy, **we are required to give you a 0** unless you **type your name** after the following statement:

> *I certify on my honor that I have neither given nor received any help, or used any non-permitted resources, while completing this evaluation.*

DAVID NGUYEN

Permitted Resources

This exam is open-book, open-notes, open-internet, open-everything. The only thing you are not allowed to do is ask another human being for help, with the exception of me. Examples:

1. **Permitted**: Checking course notes and videos; browsing Google or StackOverflow; consulting previous Campuswire posts.
2. **Not permitted**: Requesting help on Chegg; publicly posting on Campuswire (messages to me are ok); privately asking classmates for help; posting on StackOverflow.

If you encounter any situation in which you're not 100% sure what's permitted, **just ask**!

Partial Credit

Let us give you partial credit! If you're stuck on a problem and just can't get your code to run:

First, **breathe**. Then, do any or all of the following:

1. Write down everything relevant that you know about the problem, as comments where your code would go.
2. If you have non-functioning code that demonstrates some correct ideas, indicate that and keep it (commented out).
3. Write down pseudocode (written instructions) outlining your solution approach.

In brief, even if you can't quite get your code to work, you can still **show us what you know.**

Introduction

Here is a list of all of the presidents of the United States as tuples in the format `(first_name,last_name)`. (Middle initials are added to last names occasionally if needed to distinguish between father/son pairs.) For example, the tuple `("Thomas", "Jefferson")` corresponds to president Thomas Jefferson.

In [1]:
```
presidents = [("George","Washington"),
              ("John","Adams"),
              ("Thomas","Jefferson"),
              ("James", "Madison"),
              ("James", "Monroe"),
              ("John", "Q. Adams"),
              ("Andrew","Jackson"),
              ("Martin","Van Buren"),
              ("William","Harrison"),
              ("John","Tyler"),
              ("James","Polk"),
              ("Zachary","Taylor"),
              ("Milliard","Filmore"),
              ("Franklin","Pierce"),
              ("James","Buchanan"),
              ("Abraham","Lincoln"),
              ("Andrew","Johnson"),
              ("Ulysses","Grant"),
              ("Rutherford","Hayes"),
              ("James","Garfield"),
              ("Chester","Arthur"),
              ("Grover","Cleveland"),
              ("Benjamin","Harrison"),
              ("Grover","Cleveland"),
              ("William","McKinley"),
              ("Theodore","Roosevelt"),
              ("William","Taft"),
              ("Woodrow","Wilson"),
              ("Warren","Harding"),
              ("Calvin","Coolidge"),
              ("Herbert","Hoover"),
              ("Franklin","Roosevelt"),
              ("Harry","Truman"),
              ("Dwight","Eisenhower"),
              ("John","Kennedy"),
              ("Lyndon","Johnson"),
              ("Richard","Nixon"),
              ("Gerald","Ford"),
              ("Jimmy","Carter"),
              ("Ronald","Reagan"),
              ("George", "H. W. Bush"),
              ("William", "Clinton"),
              ("George","W. Bush"),
              ("Barack","Obama"),
              ("Donald","Trump"),
              ("Joseph","Biden")
             ]
```

Part A (10 points)

Write a function called `count_distinct()`. This function should take a single argument, which you may assume to be a list. The return value of `count_distinct(L)` should be the number of *distinct* items in `L`. For example, `count_distinct([1,1,"b"])` should return `2`, since `1` and `"b"` are the `2` distinct values in this list.

Specs

- **Comments and docstrings are not required for this problem,** although they may help us give you partial credit.
- You are not required to perform any type-checking for this problem.
- For full credit your solution should be **three lines or less.**

In [146]:
```
# your function definition here
def count_distinct(myList):
    return(len(set(myList)))
    #set() takes in a list and then outputs an object with items that are unordered and distinct
    #set will get rid of duplicates and len will get us the value of distinct items
```

Next, use `count_distinct()` to determine the number of different people who have been president of the United States? (Note, at least one person is on the list twice)

In [147]:
```
# test count_distinct() here
count_distinct(presidents)
```

Out[147]: 45

Part B (10 points)

Write a function called `first_names_dict` which turns inputs a list of names (stored as tuples similar to the `presidents` list above) and returns a dictionary where the keys are first names and values are the number of times that someone with that first name appears on the list.

Demonstrate your function on the list `presidents` and print the resulting dictionary.

Specs

- **Comments and docstrings are not required for this problem,** although they may help us give you partial credit.
- You are not required to perform any type-checking for this problem: it's ok to assume that the input is a correctly-constructed list whose elements are 2-tuples of strings.
- My solution has 8 lines but it is okay if yours is longer or shorter.

In [125]:
```
# define first_names_dict() here
def first_names_dict(myList):
    #get first names, they are index 0 in each name tuple
    firstNames = [name[0] for name in presidents]
    firstNameDict = dict() #declare a dict

    #for loop to go through firstNames list
    for name in firstNames:
        if name not in firstNameDict.keys():
            #set value to one if name not in dict
            firstNameDict[name] = 1
        elif name in firstNameDict.keys():
            #incrememtn if already in dict
            firstNameDict[name] += 1
    return firstNameDict
```

In [126]:
```
# demonstrate first_names_dict() here
first_names_dict(presidents)
```

Out[126]:
```
{'George': 3,
 'John': 4,
 'Thomas': 1,
 'James': 5,
 'Andrew': 2,
 'Martin': 1,
 'William': 4,
 'Zachary': 1,
 'Milliard': 1,
 'Franklin': 2,
 'Abraham': 1,
 'Ulysses': 1,
 'Rutherford': 1,
 'Chester': 1,
 'Grover': 2,
 'Benjamin': 1,
 'Theodore': 1,
 'Woodrow': 1,
 'Warren': 1,
 'Calvin': 1,
 'Herbert': 1,
 'Harry': 1,
 'Dwight': 1,
 'Lyndon': 1,
 'Richard': 1,
 'Gerald': 1,
 'Jimmy': 1,
 'Ronald': 1,
 'Barack': 1,
 'Donald': 1,
 'Joseph': 1}
```

Part C (25 points)

Write a function called `most_common_first_name()` which determines the most common first name in a list formatted in the same way as `presidents`. Your function should return both the most common first name and the number of times it appears.

In the U.S. `presidents` list, the most common first name is `"James"`, which appears 5 times. So, your function should be able to do this:

```
first_name, num_occurrences = most_common_first_name(presidents)
first_name, num_occurrences

("James", 5)
```

Specs

- **Comments and docstrings are not required for this problem,** although they may help us give you partial credit.
- You are not required to perform any type-checking for this problem: it's ok to assume that the input is a correctly-constructed list whose elements are 2-tuples of strings.
- You may assume that there is a unique most common first name. In a hypothetical list in which two names were equally common, it would be sufficient for your function to return one of them (with the correct count).

Write python code which determines what the most common first name amongst U.S. presidents is. Print the most common name and how many times it's been used when you are done. (You don't have to worry about ties. No comments are necessary for this part, but adding them could help me give you partial credit. My solution was 6 lines.)

In [148]:
```
# write your function here
def most_common_first_name(myList):
    #get dict using previous function and store in variable
    nameDict = first_names_dict(myList)
    #check dict values for highest number (most common first name)
    nameCount = max(nameDict.values())
    #find key values (first name(s)) corresponding to the highest dict value
    name = [key for key, value in nameDict.items() if value == nameCount]

    return name[0], nameCount
    #assuming there is only 1 name, we only need index 0 of name
```

In [149]:
```
# demonstrate (using the test code from the problem prompt) here
first_name, num_occurrences = most_common_first_name(presidents)
first_name, num_occurrences
```

Out[149]:
```
('James', 5)
```

In [ ]:
```

```