# Lab 4: Simulations

Welcome to Lab 4!

We will go over iteration and simulations, as well as the concept of randomness.

First, set up the tests and imports by running the cell below.

In [4]:

```
# Run this cell, but please don't change it.

# These lines import the Numpy and Datascience modules.
import numpy as np
from datascience import *

# These lines do some fancy plotting magic
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

## 1. Nachos and Conditionals

In Python, the boolean data type contains only two unique values: `True` and `False`. Expressions containing comparison operators such as `<` (less than), `>` (greater than), and `==` (equal to) evaluate to Boolean values. A list of common comparison operators can be found below!

Run the cell below to see an example of a comparison operator in action.

In [5]:

```
3 > 1 + 1
```

Out[5]:

```
True
```

We can even assign the result of a comparison operation to a variable.

In [6]:

```
result = 10 / 2 == 5
result
```

Out[6]:

```
True
```

Arrays are compatible with comparison operators. The output is an array of boolean values.

In [7]:

```
make_array(1, 5, 7, 8, 3, -1) > 3
```

Out[7]:

```
array([False,  True,  True,  True, False, False])
```

One day, when you come home after a long week, you see a hot bowl of nachos waiting on the dining table! Let's say that whenever you take a nacho from the bowl, it will either have only **cheese**, only **salsa**, **both** cheese and salsa, or **neither** cheese nor salsa (a sad tortilla chip indeed).

Let's try and simulate taking nachos from the bowl at random using the function, `np.random.choice(...)`.

## `np.random.choice`

`np.random.choice` picks one item at random from the given array. It is equally likely to pick any of the items. Run the cell below several times, and observe how the results change.

In [8]:

```
nachos = make_array('cheese', 'salsa', 'both', 'neither')
np.random.choice(nachos)
```

Out[8]:

```
'cheese'
```

To repeat this process multiple times, pass in an int `n` as the second argument to return `n` different random choices. By default, `np.random.choice` samples **with replacement** and returns an *array* of items.

Run the next cell to see an example of sampling with replacement 10 times from the `nachos` array.

In [9]:

```
np.random.choice(nachos, 10)
```

Out[9]:

```
array(['neither', 'both', 'neither', 'cheese', 'both', 'cheese', 'salsa',
       'neither', 'cheese', 'both'], dtype='<U7')
```

To count the number of times a certain type of nacho is randomly chosen, we can use `np.count_nonzero`

## `np.count_nonzero`

`np.count_nonzero` counts the number of non-zero values that appear in an array. When an array of boolean values are passed through the function, it will count the number of `True` values (remember that in Python, `True` is coded as 1 and `False` is coded as 0.)

Run the next cell to see an example that uses `np.count_nonzero`.

In [10]:

```
np.count_nonzero(make_array(True, False, False, True, True))
```

Out[10]:

```
3
```

**Question 1.1.** Assume we took ten nachos at random, and stored the results in an array called `ten_nachos` as done below. Find the number of nachos with only cheese using code (do not hardcode the answer).

*Hint:* Our solution involves a comparison operator (e.g. `=`, `<`, ...) and the `np.count_nonzero` method.

In [11]:

```
ten_nachos = make_array('neither', 'cheese', 'both', 'both', 'cheese', 'salsa',
number_cheese = np.count_nonzero(ten_nachos == 'cheese')
number_cheese
```

Out[11]:

```
3
```

In [12]:

```
# TEST
number_cheese == 3
```

Out[12]:

```
True
```

### Conditional Statements

A conditional statement is a multi-line statement that allows Python to choose among different alternatives based on the truth value of an expression.

Here is a basic example.

```
def sign(x):
    if x > 0:
        return 'Positive'
    else:
        return 'Negative'
```

If the input `x` is greater than `0`, we return the string `'Positive'`. Otherwise, we return `'Negative'`.

If we want to test multiple conditions at once, we use the following general format.

```
if <if expression>:
    <if body>
elif <elif expression 0>:
    <elif body 0>
elif <elif expression 1>:
    <elif body 1>
...
else:
    <else body>
```

Only the body for the first conditional expression that is true will be evaluated. Each `if` and `elif` expression is evaluated and considered in order, starting at the top. As soon as a true value is found, the corresponding body is executed, and the rest of the conditional statement is skipped. If none of the `if` or `elif` expressions are true, then the `else body` is executed.

For more examples and explanation, refer to the section on conditional statements here.

**Question 1.2.** Complete the following conditional statement so that the string `'More please'` is assigned to the variable `say_please` if the number of nachos with cheese in `ten_nachos` is less than `5`.

*Hint*: You should be using `number_cheese` from Question 1.

In [13]:

```
say_please = 'More please'

if number_cheese < 5:
    say_please = 'More please'
say_please
```

Out[13]:

```
'More please'
```

In [14]:

```
# TEST
say_please == 'More please'
```

Out[14]:

```
True
```

**Question 1.3.** Write a function called `nacho_reaction` that returns a reaction (as a string) based on the type of nacho passed in as an argument. Use the table below to match the nacho type to the appropriate reaction.

*Hint:* If you're failing the test, double check the spelling of your reactions.

In [15]:

```
def nacho_reaction(nacho):
    if nacho == "cheese":
        return "Cheesy!"
    elif nacho == "salsa" :
        return "Spicy!"
    elif nacho == "both" :
        return "Wow!"
    elif nacho == "neither" :
        return "Meh."

spicy_nacho = nacho_reaction('salsa')
spicy_nacho
```

Out[15]:

```
'Spicy!'
```

In [16]:

```
# TEST
nacho_reaction('salsa') == 'Spicy!'
```

Out[16]:

```
True
```

In [17]:

```
# TEST
nacho_reaction('cheese') == 'Cheesy!'
```

Out[17]:

```
True
```

In [18]:

```
# TEST
nacho_reaction('both') == 'Wow!'
```

Out[18]:
```
True
```

In [19]:
```
# TEST
nacho_reaction('neither') == 'Meh.'
```

Out[19]:
```
True
```

**Question 1.4.** Create a table `ten_nachos_reactions` that consists of the nachos in `ten_nachos` as well as the reactions for each of those nachos. The columns should be called `Nachos` and `Reactions`.

*Hint:* Use the `apply` method.

In [20]:
```
ten_nachos_tbl = Table().with_column('Nachos', ten_nachos)
ten_nachos_reactions = ten_nachos_tbl.with_column('Reactions', ten_nachos_tbl.a
ten_nachos_reactions
```

Out[20]:

| Nachos | Reactions |
|---|---|
| neither | Meh. |
| cheese | Cheesy! |
| both | Wow! |
| both | Wow! |
| cheese | Cheesy! |
| salsa | Spicy! |
| both | Wow! |
| neither | Meh. |
| cheese | Cheesy! |
| both | Wow! |

In [21]:
```
# TEST
# One or more of the reaction results could be incorrect;
np.count_nonzero(ten_nachos_reactions.column('Reactions') == make_array('Meh.',
```

Out[21]:
```
True
```

**Question 1.5.** Using code, find the number of 'Wow!' reactions for the nachos in `ten_nachos_reactions`.

In [25]:
```
number_wow_reactions = np.count_nonzero(ten_nachos_reactions.column('Reactions'
number_wow_reactions
```

Out[25]:
```
4
```

In [26]:
```
# TEST
2 < number_wow_reactions < 6
```

Out[26]:
```
True
```

In [27]:

```
# TEST
# Incorrect value for number_wow_reactions
number_wow_reactions == 4
```

Out[27]:

```
True
```

## 2. Simulations and For Loops

Using a `for` statement, we can perform a task multiple times. This is known as iteration.

One use of iteration is to loop through a set of values. For instance, we can print out all of the colors of the rainbow.

In [28]:

```
rainbow = make_array("red", "orange", "yellow", "green", "blue", "indigo", "vic

for color in rainbow:
    print(color)
```

```
red
orange
yellow
green
blue
indigo
violet
```

We can see that the indented part of the `for` loop, known as the body, is executed once for each item in `rainbow`. The name `color` is assigned to the next value in `rainbow` at the start of each iteration. Note that the name `color` is arbitrary; we could easily have named it something else. The important thing is we stay consistent throughout the `for` loop.

In [29]:

```
for another_name in rainbow:
    print(another_name)
```

```
red
orange
yellow
green
blue
indigo
violet
```

In general, however, we would like the variable name to be somewhat informative.

**Question 2.1.** In the following cell, we've loaded the text of *Pride and Prejudice* by Jane Austen, split it into individual words, and stored these words in an array `p_and_p_words`. Using a `for` loop, assign `longer_than_five` to the number of words in the novel that are more than 5 letters long.

*Hint*: You can find the number of letters in a word with the `len` function.

In [31]:

```
austen_string = open('Austen_PrideAndPrejudice.txt', encoding='utf-8').read()
```

```
p_and_p_words = np.array(austen_string.split())

longer_than_five = 0

# a for loop would be useful here
for word in p_and_p_words:
    if len(word) > 5:
        longer_than_five = longer_than_five + 1

longer_than_five
```

Out[31]:

```
35453
```

In [32]:

```
# TEST
longer_than_five == 35453
```

Out[32]:

```
True
```

**Question 2.2.** Using a simulation with 10,000 trials, assign num_different to the number of times, in 10,000 trials, that two words picked uniformly at random (with replacement) from Pride and Prejudice have different lengths.

*Hint 1*: What function did we use in section 1 to sample at random with replacement from an array?

*Hint 2*: Remember that `!=` checks for non-equality between two items.

In [34]:

```
trials = 10000
num_different = 0

for trial in np.arange(trials):
    words = np.random.choice(p_and_p_words, 2)
    if len(words.item(0)) != len(words.item(1)):
        num_different = num_different + 1

num_different
```

Out[34]:

```
8656
```

In [35]:

```
# TEST
8100 <= num_different <= 9100
```

Out[35]:

```
True
```

We can also use `np.random.choice` to simulate multiple trials.

**Question 2.3.** Allie is playing darts. Her dartboard contains ten equal-sized zones with point values from 1 to 10. Write code that simulates her total score after 1000 dart tosses.

*Hint:* First decide the possible values you can take in the experiment (point values in this case). Then use `np.random.choice` to simulate Allie's tosses. Finally, sum up the scores to get Allie's total score.

In [39]:

```
possible_point_values = make_array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
num_tosses = 1000
simulated_tosses = np.random.choice(possible_point_values, num_tosses)
total_score = sum(simulated_tosses)
total_score
```

Out[39]:

```
5356
```

In [40]:

```
# TEST
1000 <= total_score <= 10000
```

Out[40]:

```
True
```

# 3. Probability

We will be testing some probability concepts that were introduced in lecture. For all of the following problems, we will introduce a problem statement and give you a proposed answer. You must assign the provided variable to one of the following three integers, depending on whether the proposed answer is too low, too high, or correct.

1. Assign the variable to 1 if you believe our proposed answer is too high.
2. Assign the variable to 2 if you believe our proposed answer is too low.
3. Assign the variable to 3 if you believe our proposed answer is correct.

You are more than welcome to create more cells across this notebook to use for arithmetic operations

**Question 3.1.** You roll a 6-sided die 10 times. What is the chance of getting 10 sixes?

Our proposed answer:

$$\left(\frac{1}{6}\right)^{10}$$

Assign `ten_sixes` to either 1, 2, or 3 depending on if you think our answer is too high, too low, or correct.

In [41]:

```
ten_sixes = 3
ten_sixes
```

Out[41]:

```
3
```

In [42]:

```
# TEST
ten_sixes == 3
```

Out[42]:

```
True
```

**Question 3.2.** Take the same problem set-up as before, rolling a fair dice 10 times. What is the chance that every roll is less than or equal to 5?

Our proposed answer:

$$1 - \left(\frac{1}{6}\right)^{10}$$

Assign `five_or_less` to either 1, 2, or 3.

In [43]:

```
five_or_less = 1
five_or_less
```

Out[43]:

```
1
```

In [44]:

```
# TEST
five_or_less == 1
```

Out[44]:

```
True
```

**Question 3.3.** Assume we are picking a lottery ticket. We must choose three distinct numbers from 1 to 1000 and write them on a ticket. Next, someone picks three numbers one by one from a bowl with numbers from 1 to 1000 each time without putting the previous number back in. We win if our numbers are all called in order.

If we decide to play the game and pick our numbers as 12, 140, and 890, what is the chance that we win?

Our proposed answer:

$$\left(\frac{3}{1000}\right)^{3}$$

Assign `lottery` to either 1, 2, or 3.

In [45]:

```
lottery = 1
```

In [46]:

```
# TEST
lottery == 1
```

Out[46]:

```
True
```

**Question 3.4.** Assume we have two lists, list A and list B. List A contains the numbers [20,10,30], while list B contains the numbers [10,30,20,40,30]. We choose one number from list A randomly and one number from list B randomly. What is the chance that the number we drew from list A is larger than or equal to the number we drew from list B?

Our proposed solution:

$$1/5$$

Assign `list_chances` to either 1, 2, or 3.

*Hint: Consider the different possible ways that the items in List A can be greater than or equal to items in List B. Try working out your thoughts with a pencil and paper, what do you* *think the correct solutions will be close to?*

In [47]:

```
list_chances = 2
```

In [48]:

```
# TEST
list_chances == 2
```

Out[48]:

```
True
```

Great job! You're finished with lab 4! Be sure to...

- **run all the tests**,
- **print the notebook as a PDF**,
- and **submit both the notebook and the PDF to Canvas**.