Section 4.6 Exercise #1

Define a function in a Jupyter notebook cell that accepts one parameter and prints that parameter three times.

In a different Jupyter notebook cell, call the function several different times with different arguments.

In [3]:
```
def print_three_times(print_this):
    """Prints the function's input three times.

    Args:
        print_this: a string.

    Returns:
        Nothing
    """
    for idx in range(3):
        print(print_this)
```

In [4]:
```
print_three_times("Functions are rad.")
```

```
Functions are rad.
Functions are rad.
Functions are rad.
```

Section 4.6 Exercise #2

Define and run a function with two parameters.

- The function should add the two parameters and return the sum.
- Call the function and save the return valuable in a variable
- Print the variable that contains the function's return value

In [5]:
```
def add_two_numbers(param1, param2):
    """Adds two numbers.

    Args:
        param1: a number
        param2: another number

    Returns:
        The sum of param1 and param2
    """
    return param1 + param2
```

In [6]:
```
sum = add_two_numbers(13, 18)
print(sum)
```

```
31
```

Section 4.7 Exercise #1

Define a function that multiplies two numbers and returns the product.

- The function should accept either one or two arguments.
- If the function is called with two arguments, the function should return the product of the two arguments
- If the function is called with one argument, it should return that argument multiplied by 1318.

In [7]:
```
def multiply_stuff(param1, param2=1318):
    """Multiplies two numbers.

    If only one argument is provided, the argument will be multiplied
    by 1,318.

    Args:
        param1: a number
        param2: another number. Optional. Defaults to 1318.

    Returns:
        If two arguments are provided, returns the product of the two
        numbers.

        If one argument is provided, returns the product of the
        argument with 1,318.
    """
    return param1 * param2
```

In [8]:
```
multiply_stuff(2, 3)
```

Out[8]:
```
6
```

In [9]:
```
multiply_stuff(2)
```

Out[9]:
```
2636
```

Section 4.7.2 Exercise #1

Write a function that accepts two arguments.

- The function should divide the first argument by the second argument and return the quotient
- Call the function using positional arguments.
- Call the function using keyword arguments, with the 2nd argument placed before the 1st argument in the function call

In [10]:
```
def divide_stuff(dividend, divisor):
    """Divides one argument by another.

    Args:
        dividend: a number.
        divisor: another number.

    Returns:
        The quotient of dividen/divisor
    """
    return dividend / divisor
```

In [11]:
```
divide_stuff(10, 5)
```

Out[11]: 2.0

In [12]:
```
divide_stuff(5, 10)
```

Out[12]: 0.5

In [13]:
```
divide_stuff(divisor=5, dividend=10)
```

Out[13]: 2.0

Function Ex. #1

Write a function that calculates the area of a circle given the radius.

In [8]:
```
import math

def area_circ(radius):
    """Calculates the radius of a circle.

    Args:
        radius: a number.

    Returns:
        A float value that is the area of a circle with the given
        radius.
    """
    return math.pi * radius**2
```

In [10]:
```
area_circ(2)
```

Out[10]: 12.566370614359172

In [14]:
```
area_circ(1/math.sqrt(math.pi))
```

Out[14]: 0.9999999999999999

Function Ex. #2

Write a function that takes the number of teams at a competition and the number of matches per team and calculates the number of qualification matches that will be needed

In [2]:
```
def calc_num_matches(num_teams, matches_per_team):
    """Determines the number of qual matches needed in an FRC competition.

    Args:
        num_teams: An integer representing the number of teams at an
            FRC competition.
        matches_per_team: The number of qualificaiton matches that each
            individual team will play at a the compeition.

    Returns:
        The number of qualification matches needed. If the return value
        is an integer, no surrogates will be needed.
    """
    # How many matches needed if only one team in every match?
    team_matches = num_teams * matches_per_team

    # But there are six teams in every match!
    return team_matches / 6
```

In [3]:
```
calc_num_matches(num_teams=35, matches_per_team=12)
```

Out[3]:
```
70.0
```

In [6]:
```
calc_num_matches(num_teams=62, matches_per_team=9)
```

Out[6]:
```
93.0
```

In [9]:
```
calc_num_matches(num_teams=68, matches_per_team=10)
```

Out[9]:
```
113.33333333333333
```

Advanced Ex. #1

Write a function that takes arguments and extracts data from the JSON used in session 2, does calculations, and returns a value. You choose what the function does.

In [42]:
```
import json

def breaks(team):
    """For one team, gets the number of matches between each assigned match.

    For any two consecutive matches that a team plays in, gets the number
    of matches between the two matches that the team does not play in.
    For example, if a team is assigned to qual matches 2 and 4, breaks()
    returns 1 (there is 1 match between matches 2 and 4 that the team
    does not play in).

    Args:
        team: FRC team number as an integer, such as 1318, or 2557.

    Returns:
        A list of integers representing the breaks in the schedule
        for the specified team.
    """
    sched_file = open('sched_turing_2018.json')
    sched_data = json.load(sched_file)
    sched_file.close()

    matches = [None] * 10
    idx = 0
    for mtch in sched_data['Schedule']:
        for tm in mtch['teams']:
            if tm['teamNumber'] == team:
                matches[idx] = mtch['matchNumber']
                idx += 1

    breaks = [None] * 9
    idx = 0
    prev_mtch = matches[0]
    for curr_mtch in matches[1:]:
        breaks[idx] = curr_mtch - prev_mtch - 1
        prev_mtch = curr_mtch
        idx += 1
    return breaks
```

In [44]:
```
breaks(1318)
```

Out[44]:
```
[9, 11, 9, 10, 18, 11, 8, 5, 19]
```

Advanced Function Ex. #2

Write a function that calculates the probability that in a group of N people, no two people will have the same birthday. (Ignore leap year and assume that all days of the year have equal probability for being someone's birthday.)

In [18]:
```
import math

def bday(num_people):
"""Calculates probability that no two people will have the same birthday.

Args:
    num_people: the number of people in the group (integer)

Returns:
    The probability (float between 0 and 1) that no two people in the group
    have the same birthday.
"""
    unique_combinations = math.factorial(365)/math.factorial(365 - num_people)
    all_possible_combinations = 365**num_people

    return unique_combinations / all_possible_combinations
```

In [20]:
```
print(bday(23))
```

```
0.4927027656760146
```

Advanced Function Ex. #3

Write a function that takes the playoff alliance numbers as arguments (e.g., alliance #1, alliance #5, etc) and returns a string indicating the level at which those alliances would meet in the playoffs (e.g., finals, quarterfinals, etc.). This is a tricky one. *See page 104 of the Deep Space game manual if you are not sure what this exercise is asking for.*

In [25]:
```python
def playoff_lvl(al1, al2):
    """Determines when two alliances will meet in the playoffs.

    Args:
        al1: The number of the first alliance, an integer ranging
            form 1 to 8.
        al2: The number of the second alliance, an integer ranging
            form 1 to 8.

    Returns:
        A string. Either 'never', 'quarter-finals', 'semi-finals',
        of 'finals'.

    """
    # Must enter two different alliance numbers
    if al1 == al2:
        return 'never'

    # Check for quarterfinal matchups
    for qf_mtch in [[1, 8], [4, 5], [2, 7], [3, 6]]:
        if al1 in qf_mtch and al2 in qf_mtch:
            return "quarter-finals"

    # Check for semifinal and final matchups
    sf_red = [1, 8, 4, 5]
    sf_blue = [2, 7, 3, 6]
    al1_sf = 'red' if al1 in sf_red else 'blue'
    al2_sf = 'red' if al2 in sf_red else 'blue'
    if al1_sf == al2_sf:
        return 'semi-finals'
    else:
        return 'finals'
```

In [24]:
```python
playoff_lvl(1, 4)
```

Out[24]:
```
'semi-finals'
```

Python Style Example

In [37]:
```
def odd_or_even(input = None):

    """The function odd_or_even determines if a
    number is odd or even.
    Args: (int) input, an integer
    """
    print("We will now use the power of modern technology " + "to do something
a seven-year-old can do with no trouble.");
    print() ; print()
    if (input is None) :
        print( "Try entering a number next time!")
    elif not isinstance(input,int):
      print("'{}' isn't an integer. Try again.".format( input))
    elif input%2 == 0:
        print(str( input) + " is even.");
    else:
        print("{} is odd.".format(input))
```

In [38]:
```
odd_or_even()
```

```
We will now use the power of modern technologyto do something a seven-year-ol
d can do with no trouble.


Try entering a number next time!
```

In [39]:
```
odd_or_even("one")
```

```
We will now use the power of modern technologyto do something a seven-year-ol
d can do with no trouble.


'one' isn't an integer. Try again.
```

In [40]:
```
odd_or_even(1)
```

```
We will now use the power of modern technologyto do something a seven-year-ol
d can do with no trouble.


1 is odd.
```

In [41]:
```
odd_or_even(2)
```

```
We will now use the power of modern technologyto do something a seven-year-ol
d can do with no trouble.


2 is even.
```

In [ ]: