PART 1: PYTHON - A (VERY) SHORT INTRODUCTION

1.1 VARIABLES

TASK: Define variables and write code to...

1. Calculate square root of 25, 144 and 289
2. Translate the following into Python
    - assign numbers between 1 and 100 to 3 different variables
    - add up your 3 variables and print whether the sum is bigger than 100

Hints:

- You can perform clever exponentiation (what does x^0.5 do to x?)
- look at the section on logical operators if you're stuck!

In [2]:

```
# Your code for example 1 here
print(25**0.5)
print(144**.5)
print(289**.5)
```

```
5.0
12.0
17.0
```

In [3]:

```
# Your code for example 2 here
var1 = 20
var2 = 66
var3 = 2
print((var1 + var2 + var3) > 100)
```

```
False
```

1.2 LOOPS and STATEMENTS

TASK: Write your own for loop and if/then statement

Given

```
modules = ["cognitive", "developmental", "neuro", "clinical", "social",
"computational"]
comparisonVariable = "introductory"
```

Determine whether any of the modules have names that are longer than the comparison variable.

In [1]:

```
modules = ["cognitive", "developmental", "neuro", "clinical", "social", "computa
tional"]
comparisonVariable = "introductory"

# Your code here
anyNameLonger = False
for mod in modules:
    if len(mod) > len(comparisonVariable):
        anyNameLonger = True
print(anyNameLonger)
```

```
True
```

1.3 FUNCTIONS

Task: Write your own function!

Write a function that takes a list of numbers as inputs and returns the mean of these numbers!
Remember, given a vector of numbers

$$\mathbf{x} = [x_1, x_2, x_3, x_4, \ldots, x_n]$$

the mean is defined as:

$$mean(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{1}{N}(x_1 + x_2 + x_3 + \ldots + x_n)$$

In [5]:

```
numbers = [2,5,8,7,1,4,4,9]
# your code goes here
def compute_mean(x):
    return sum(x)/len(x)
print(compute_mean(numbers))
```

```
5.0
```

Now write a function that computes the variance of a list of numbers. Remember, the variance is defined as

$var(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$, where $mean(\mathbf{x}) = \bar{x}$

or alternatively:

$var(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}(x_i^2) - \bar{x}^2$ ("expectation of square minus square of expectation")

In [6]:

```
numbers = [2,5,8,7,1,4,4,9]
# your code goes here
def compute_variance(x):
    return compute_mean([ii**2 for ii in x])-compute_mean(x)**2
print(compute_variance(numbers))
```

```
7.0
```

Now add docstrings to your functions to explain how to use them

In [4]:

```
#SAMPLE SOLUTIONS
def compute_mean(x):
    '''
    computes mean of a list of numbers
    INPUT: x (list)
    OUTPUT: mu (float)
    '''
    return sum(x)/len(x)

def compute_variance(x):
    '''
    computes variance of a list of numbers
    INPUT: x (list)
    OUTPUT: var (float)
    '''
    return compute_mean([ii**2 for ii in x])-compute_mean(x)**2

print(help(compute_mean))
print(help(compute_variance))
```

```
Help on function compute_mean in module __main__:

compute_mean(x)
    computes mean of a list of numbers
    INPUT: x (list)
    OUTPUT: mu (float)

None
Help on function compute_variance in module __main__:

compute_variance(x)
    computes variance of a list of numbers
    INPUT: x (list)
    OUTPUT: var (float)

None
```

1.4 .METHODS()

Task: String Manipulation

You're given the following two strings:

```
gibberish_1 = 'is much I can''t enough.'
gibberish_2 = 'cOdinG sO FuN. JuST gEt '
```

- turn the strings into lists of words (use the .split method)
- make sure that gibberish_2 contains only lower case letters (.lower() method)
- create a new empty list (assign [] to a variable)
- loop through 5 steps and for each iteration, add pairs of words to your new empty list
- turn the list into a string (use the ' '.join(list) method)
- print the result

In [55]:

```
gibberish_1 = 'is much I can''t enough.'
gibberish_2 = 'cOdinG sO FuN. JuST gEt '

# Your code goes here
# SAMPLE SOLUTION 1
seq1 = gibberish_1.split()
seq2 = gibberish_2.lower().split()
wordlist = []
for ii in range(len(seq1)):
    wordlist.append(' '.join((seq2[ii], seq1[ii])))
sentence = ' '.join(wordlist)
print(sentence)

# SAMPLE SOLUTION 2
seq1 = gibberish_1.split()
seq2 = gibberish_2.lower().split()
wordlist = []
for ii in range(len(seq1)):
    wordlist.append(seq2[ii])
    wordlist.append(seq1[ii])
sentence = ' '.join(wordlist)
print(sentence)

# SAMPLE SOLUTION 3
seq1 = gibberish_1.split()
seq2 = gibberish_2.lower().split()
print(' '.join([' '.join((seq2[ii],seq1[ii])) for ii in range(len(seq1))]))
```

```
coding is so much fun. I just cant get enough.
coding is so much fun. I just cant get enough.
coding is so much fun. I just cant get enough.
```

PART2: PYTHON FOR SCIENTISTS: PACKAGES/LIBRARIES

TASK: DATA PLOTTING

Try to plot lengths against widths of the petals for all flower types. The kind of plot you'll want to use is most likely a scatterplot, so see if you can google your way to syntax that will take petal length and width as inputs for each datapoint.

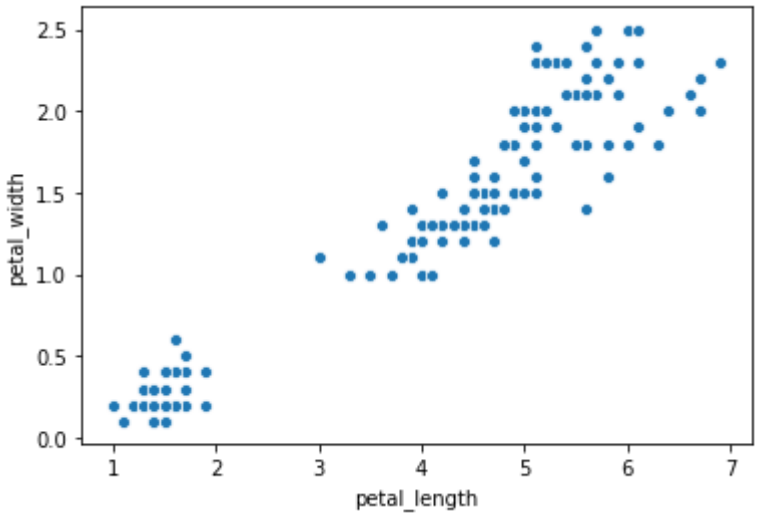
In [4]:

```
# Your code for task 1 goes here:

sns.scatterplot(iris['petal_length'],iris['petal_width'])
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb294db3048>
```



Now try to change your scatterplot code from the cell above to color different species of flowers differently. Hint: Seaborn is fancy, so it will call color "hue"... look for hue designations in scatterplot documentation for help.

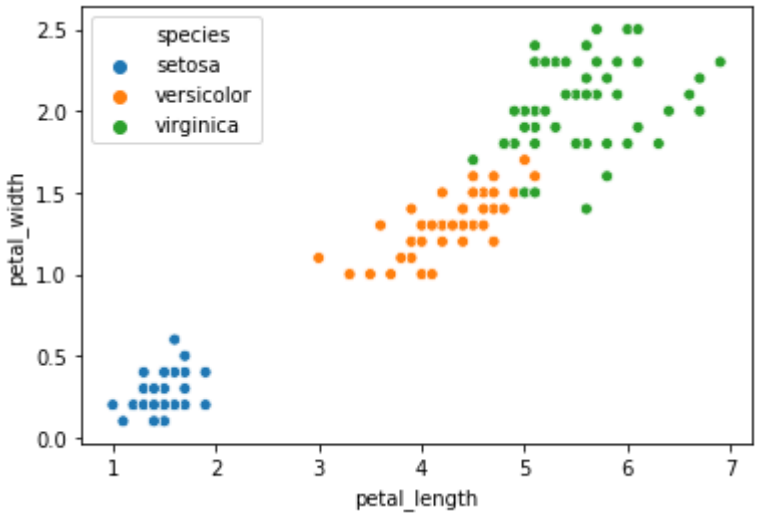
In [3]:

```
# Your code goes here


import seaborn as sns
sns.scatterplot(iris['petal_length'],iris['petal_width'],hue=iris['species'])
```

Out[3]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb29771eb70>
```



TASK: DATA VISUALIZATIONS

1. Try to plot all datapoints of petal lenghts by species in this dataset. What you want are little clusters of petal length values distributed by type of flower in your dataset. Hint: you might want to look at **swarmplots** using seaborn for this.

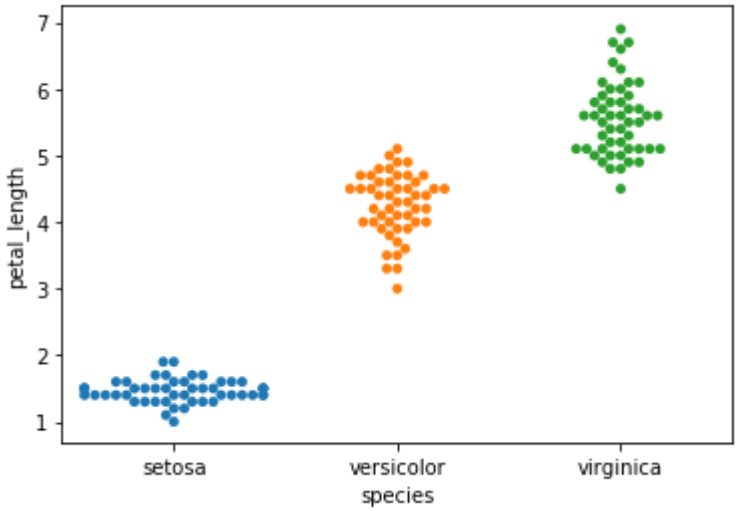
In [6]:

```
# Your code for task 1 goes here:

sns.swarmplot(x=iris['species'],y=iris['petal_length'])
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb29247b4e0>
```



1. When you are done with 1, try to plot means and standard deviations of petal lengths by species. Essentially, you are trying to create the exact same visualization as in the previous task, but now instead of seeing every data point you are interested in seeing the key values of your set - mean and stdev. Hint: you might want to look at **boxplots** using seaborn for this.

In [7]:

```
# Your code for task 2 goes here:

sns.boxplot(x=iris['species'],y=iris['petal_length'])
```

Out[7]:

<matplotlib.axes._subplots.AxesSubplot at 0x7fb2906575f8>