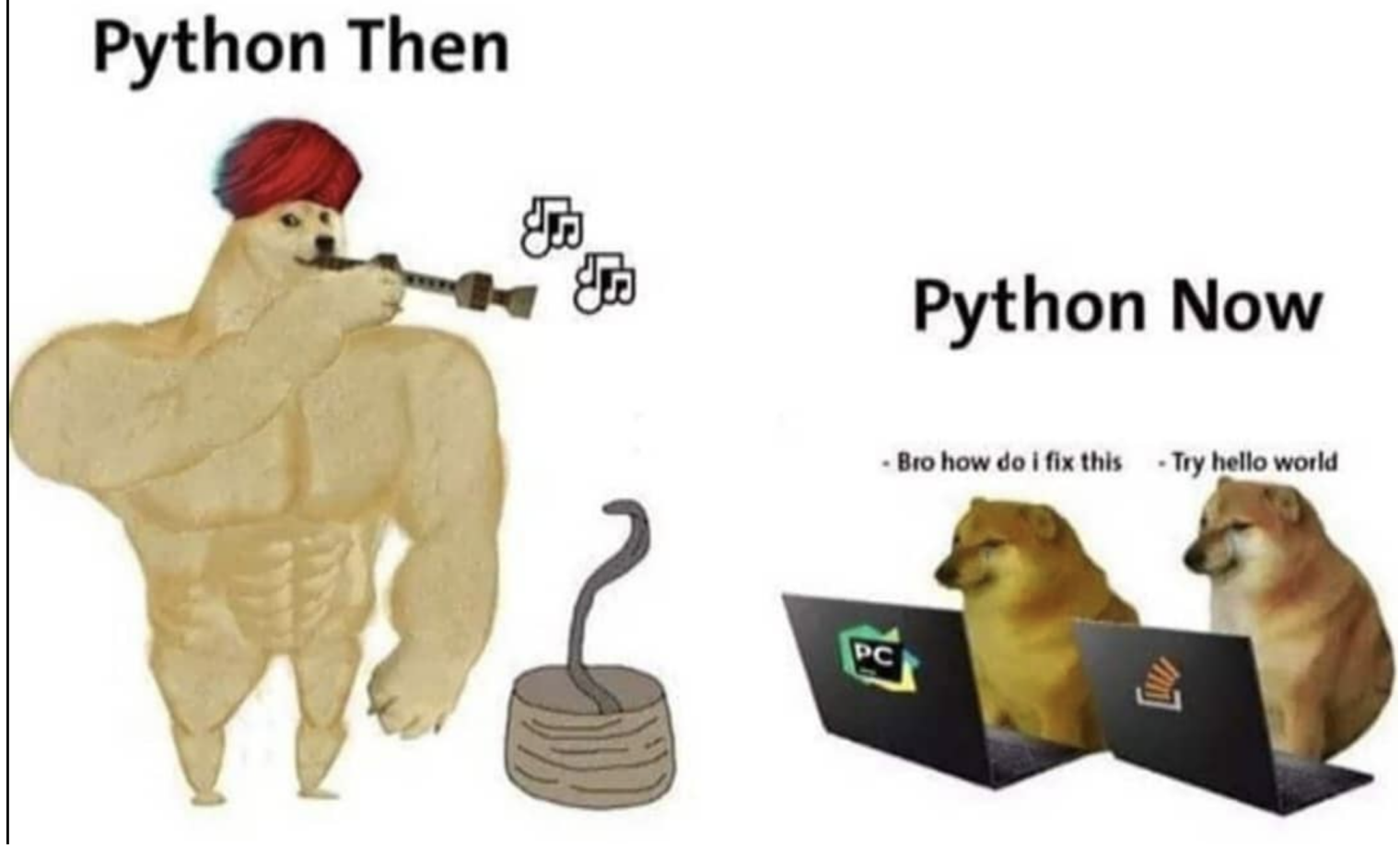
Discussion 5

DSC 20, Winter 2024

Meme of the week



Agenda

- Skill test
- **Content**
    - HOF
    - Time Complexity
- (if we have time) project foreshadowing: machine learning
- Practice Questions

Skill 1 (Read Existing File)

Given an existing, non-empty file, write the function get_line that returns the content on the last line of the file. You may assume that the last line of the file has no new line character attached at the end.

In [1]:
```
def get_line(file_path):
    """
    >>> get_line('infile.txt')
    'wassup'
    """
    with open(file_path, 'r') as f:
        return f.readlines()[-1]
```

In [ ]:
```
get_line('files/infile.txt')
```

Skill 2 (Write a File)

Given a file path and a string, implement the function write_char that writes each character in the given string to the file, in order and one on each line. The file should end with the newline character.

In [11]:
```
def write_char(file_path, string):
    """
    >>> write_char('out.txt', 'abcd')
    >>> with open('out.txt', 'r') as f:
    ...     content = f.read()
    >>> print(content)
    a
    b
    c
    d
    <BLANKLINE>
    """
    with open(file_path, 'w') as f:
        data = list(string)
        f.write('\n'.join(data))
        f.write('\n')
```

In [ ]:
```
write_char('files/out.txt', 'abcd')
with open('files/out.txt', 'r') as f:
     print(f.read())
```

Skill 3 (Append)

Given an input file, a string, a start index and an end index, implement the function append_char that writes a function that appends the characters starting from the start index (inclusive) to the end index (inclusive) on one new line to the input file. The file should end with the newline character.

In [27]:
```
def append_char(file_path, string, start_idx, end_idx):
    """
    >>> append_char('task3.txt', 'dsc!!', 0, 2)
    >>> with open('task3.txt', 'r') as f:
    ...     content = f.read()
    >>> print(content)
    this class is fun
    dsc
    <BLANKLINE>
    """
    with open(file_path, 'a') as f:
        f.write('\n')
        f.write(string[start_idx:end_idx+1])
        f.write('\n')
```

In [ ]:
```
append_char('files/task3.txt', 'dsc!!', 0, 2)
with open('files/task3.txt', 'r') as f: print(f.read())
```

Skill 4 (Read then Write)

Write a function read_write that reads the content from an input file and writes each line in reverse (as in reverse all the characters in the line) to a new file. The order of the lines should be the same as they are in the input file. The file should end with a new line.

In [33]:
```
def read_write(read_file, write_file):
    """
    >>> read_write('task4_input.txt', 'task4_output.txt')
    >>> with open('task4_output.txt', 'r') as f:
    ...     content = f.read()
    >>> print(content)
    you are almost done
    this class is fun
    i love dsc
    <BLANKLINE>
    """
    with open(read_file ,'r') as f:
        data = f.readlines()
        with open(write_file, 'w') as g:
            for line in data:
                g.write(line.strip()[::-1])
                g.write('\n')
```

In [ ]:
```
read_write('files/task4_input.txt', 'files/task4_output.txt')
with open('files/task4_output.txt', 'r') as f: print(f.read())
```

HOF example: summation

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

In [11]:
```
def summation_i(n):
    return (n*(n+1)) / 2

def summation_i2(n):
    return (n*(n+1)*(2*n+1)) / 6

def summation_formulas(n, form):
    if form=='i':
        return summation_i(n)
    if form=='i^2':
        return summation_i2(n)

summation_formulas(4, 'i^2')
```

Out[11]: 30.0

HOF (cont.)

In [13]:
```
def add_1(x):
    return x + 1
def minus_1(x):
    return x - 1

def operate(op_type):
    if op_type == 'add':
        return add_1
    else:
        return minus_1
operate('add')
```

Out[13]:
```
<function __main__.add_1(x)>
```

Checkpoint

Given the previous code, what is the result of the function call?

In [31]:
```
def add_1(x): # code from last slide
    return x + 1
def minus_1(x):
    return x - 1

def operate(op_type):
    if op_type == 'add':
        return add_1
    else:
        return minus_1
```

In [ ]:
```
operate('add')(1)
```

A. 1

B. 2

C. <function **main**.add_1(x)>

D. Error

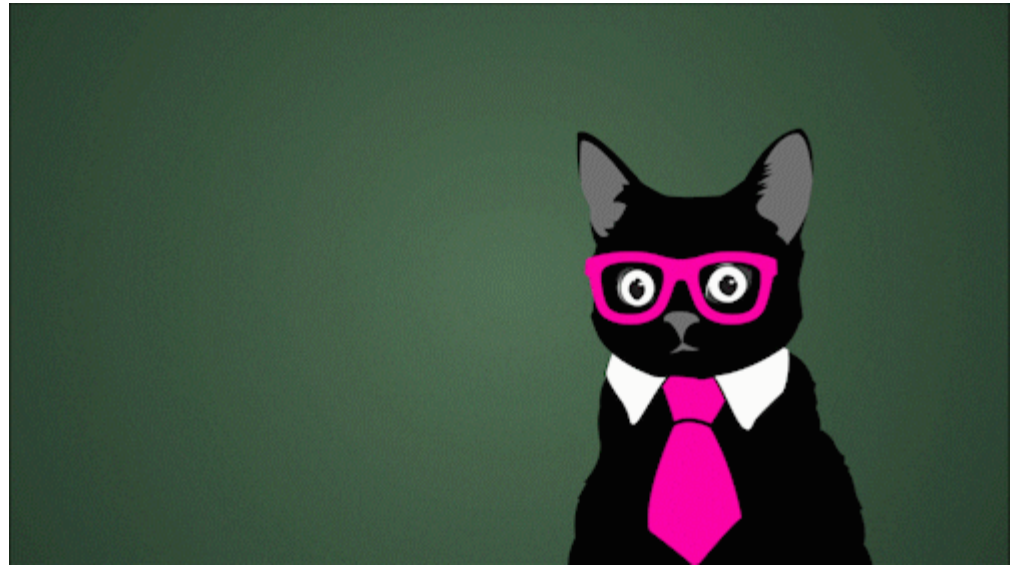
Checkpoint Solution

In [17]:
```
operate('add')(1) # B
```

Out[17]:
```
2
```

Complexity

Time complexity is an empirical method to measure the efficiency of code. Since everyone's computer is different, we can't compare code with actual numbers. Instead, we classify them into different runtime categories that allow us to infer its runtime relative to our input.



Complexity Fundamentals

1. Code should be quantified into big O values
2. Nested code will have compounded big O values
3. The largest term dictates the growth rate (i.e. only largest term matters)
4. Constants are irrelevant
5. big O analysis uses similar ideas to calculus and limits - reference your math knowledge

Complexity - Basic Interpretation

Operations that take a **constant** time to run and run at the same speed regardless of input size ->

In [ ]:
```
lst = [1,2,3]
len(lst) # <- len is constant!
```

A statement that takes **linear** time to run will increase linearly with the size of input ->

In [ ]:
```
sum(lst) # <- sum is linear!
```

Complexity - Basic Interpretation (Cont.)

A statement that takes **quadratic** time to run will increase quadratically with the size of input ->

In [ ]:
```
for i in range(len(lst)): # O(n^2)
    for j in range(len(lst)):
        print(i + j)
```

A statement that takes **logarithmic** time to run will increase logarithmically with the size of input ->

In [ ]:
```
i = n
while i > 0: # O(log(n))
    i = i // 2
```

Complexity - Basic Interpretation (Cont.)

These are very basic examples. Just because there's a nested for loop does not necessarily mean that the runtime is O(n^2). Runtime depends on the number of iterations happening and the cost of each calculation. For example, every function you've used in this class so far has a specific runtime -> sorting is usually $O(nlogn)$

Checkpoint

What is the runtime of the following function?

In [3]:
```
def boo(lst):
    for i in lst:
        for j in 1000:
            print(i + j)
```

A.

B.

C.

D.

E. Other

Checkpoint Solution

Solution:

The second for loop is a bait! The outer loop grows linearly with the input, but the second loop is constant.

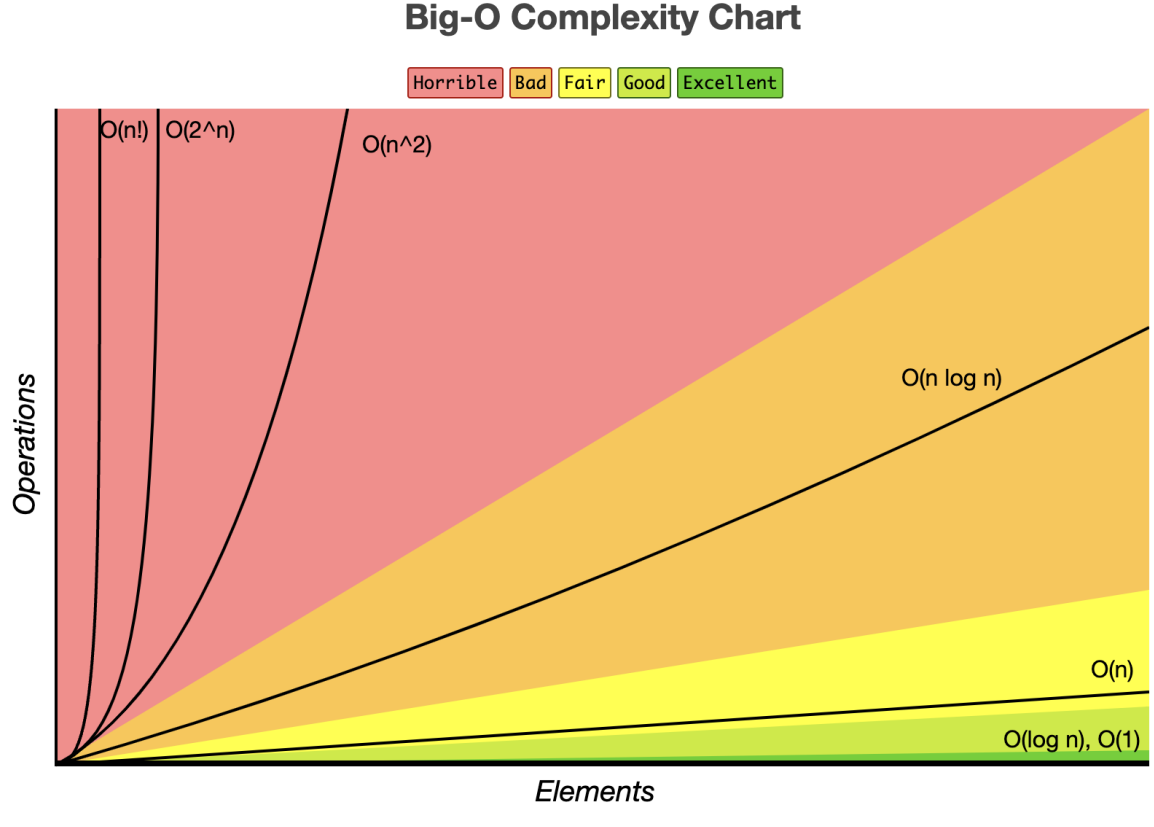
In [5]:
```
def boo(lst):
    for i in lst: # O(n)
        for j in 1000: # O(1)
            print(i + j) # O(1)
```

(boo) =

Complexity - Calculation

tips:

1. Always look for "hallmark features" (ex. if I see i // 2, there's probably log involved)
2. If there are loops, check the iteration count (don't assume n runtime)
3. Order of Growth follows mathematical principles. Only consider the largest term



Checkpoint

What is the runtime of the following function?

In [4]:
```
def bubble_sort(arr):
    """
    function that sorts a list of values.
    """
    n = len(arr)
    for i in range(n):
        for j in range(n - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

A.

B.

C.

D.

E. Other

Checkpoint Solution

Solution:

Outer for loop, which runs times. Inner for loop that runs times. Within these 2 for loops, operations are all constant.

In [ ]:
```
def bubble_sort(arr):
    n = len(arr) # O(1)
    for i in range(n): # O(n)
        for j in range(n - 1): # O(n-1)
            if arr[j] > arr[j + 1]: # O(1)
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

(bubble_sort) =

Checkpoint

What is the runtime of the following function?

In [8]:
```
def foo(lst):
    output = []
    for i in lst:
        if i%2==0:
            output.append(i)

    for i in range(len(output)):
        output[i] *= 2
    return output
```

A.

B.

C.

D.

E. Other

Checkpoint Solution

Solution:

First for loop, which runs times. Second for loop that runs times. Within these 2 for loops, operations are all constant.

In [ ]:
```
def foo(lst):
    output = []
    for i in lst: #O(n)
        if i%2==0:
            output.append(i) #O(1)

    for i in range(len(output)): #O(n)
        output[i] *= 2 #O(1)
    return output
```

(foo) =

In [38]:
```
def median(vals):
    length = len(vals)
    data = sorted(vals)
    if length%2==0:
        return (data[(length-1)//2] + data[length//2])/2
    return data[length//2]
```

Solution:

Most expensive operation is sorting - as previously stated, python's default sorting mechanisms are runtime complexity. All other operations are constant time.

In [ ]:
```
def spread(vals):
    return max(vals) - min(vals)
```

Solution:

both max and min have a runtime of , resulting in a runtime of since each are caleld once.

In [39]:

```
def summary_stat(operation):
    """
    Write a function that takes in a specified operation
    and returns a function that will take in a set of
    numbers and calculate the operation accordingly.

    possible operations:
        min -> finds the minimum value
        max -> finds the maximum value
        spread -> finds the spread of the values
        median -> finds the median of the values

    >>> med = summary_stat('median')
    >>> med([1,4,3,2,6,5])
    3.5
    >>> ran = summary_stat('range')
    >>> ran([1,4,3,2,6,5])
    5
    """
    if operation=='min':
        return min
    if operation=='max':
        return max
    if operation=='median':
        return median
    else:
        return spread
print(summary_stat('median')([1,4,3,2,6,5]))
print(summary_stat('range')([1,4,3,2,6,5]))
```

Machine Learning

Time for some actual data science!

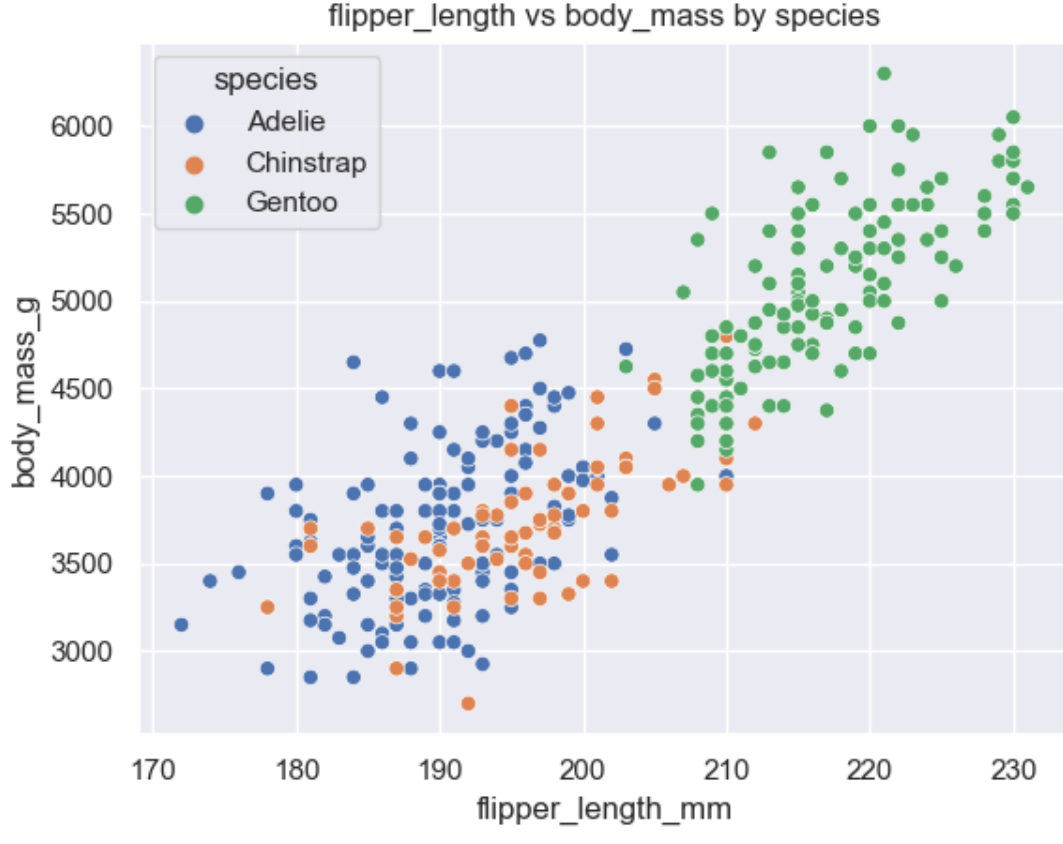
Learning always starts with data. Without having some "ground truth" to base your learning on, whatever you learn is ultimately meaningless. For today's discussion, we'll take a look at a hallmark dataset for data science, penguins:

To simplify the problem, we will not be using all features. Instead, we will only consider **'body_mass_g', 'flipper_length', and 'species'**.

	flipper_length_mm	body_mass_g	species
0	181.0	3750.0	Adelie
1	186.0	3800.0	Adelie
2	195.0	3250.0	Adelie
4	193.0	3450.0	Adelie
5	190.0	3650.0	Adelie

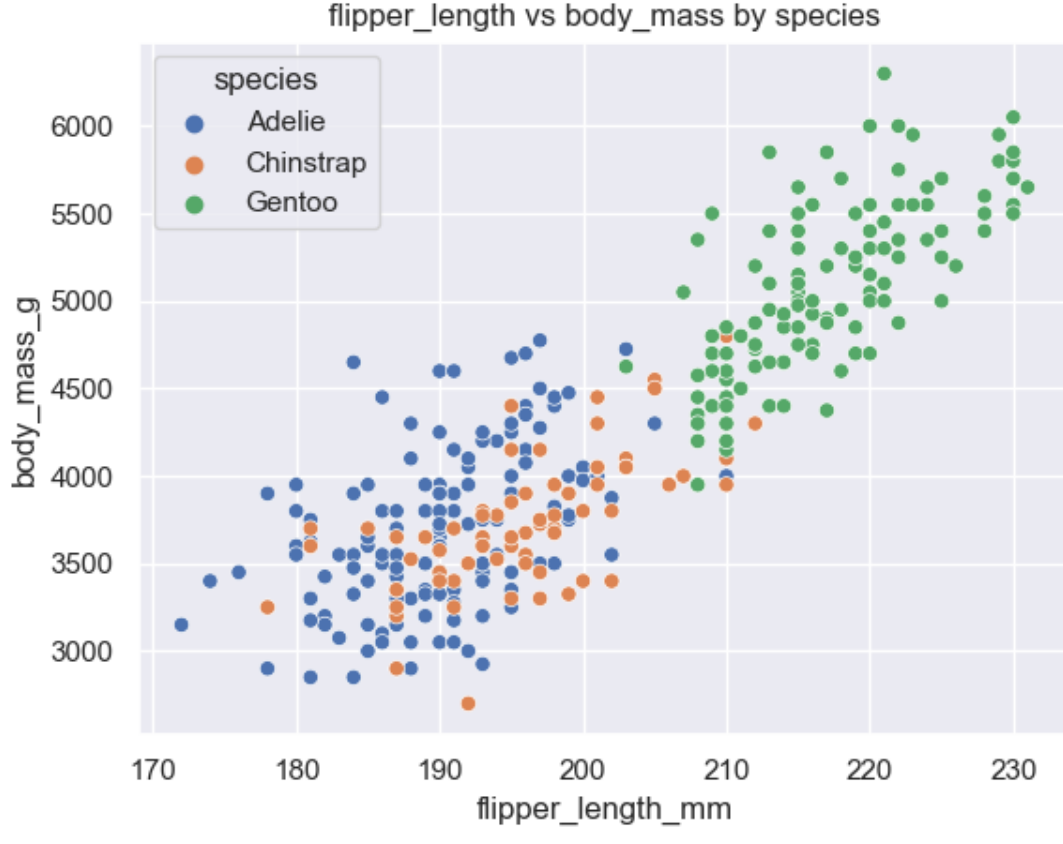
Classifiers

Classification is one of the major tasks in machine learning. Its goal (as its name implies) is to ingest data associated with a label and be able to predict the label of future unseen data by learning some sort of pattern. Within our penguins dataset, this would involve predicting the penguin species ('species') based on its 'flipper_length_mm' and 'body_mass_g'.



Classifiers

Classification is one of the major tasks in machine learning. Its goal (as its name implies) is to ingest data associated with a label and be able to predict the label of future unseen data by learning some sort of pattern. Within our penguins dataset, this would involve predicting the penguin species ('species') based on its 'flipper_length_mm' and 'body_mass_g'.

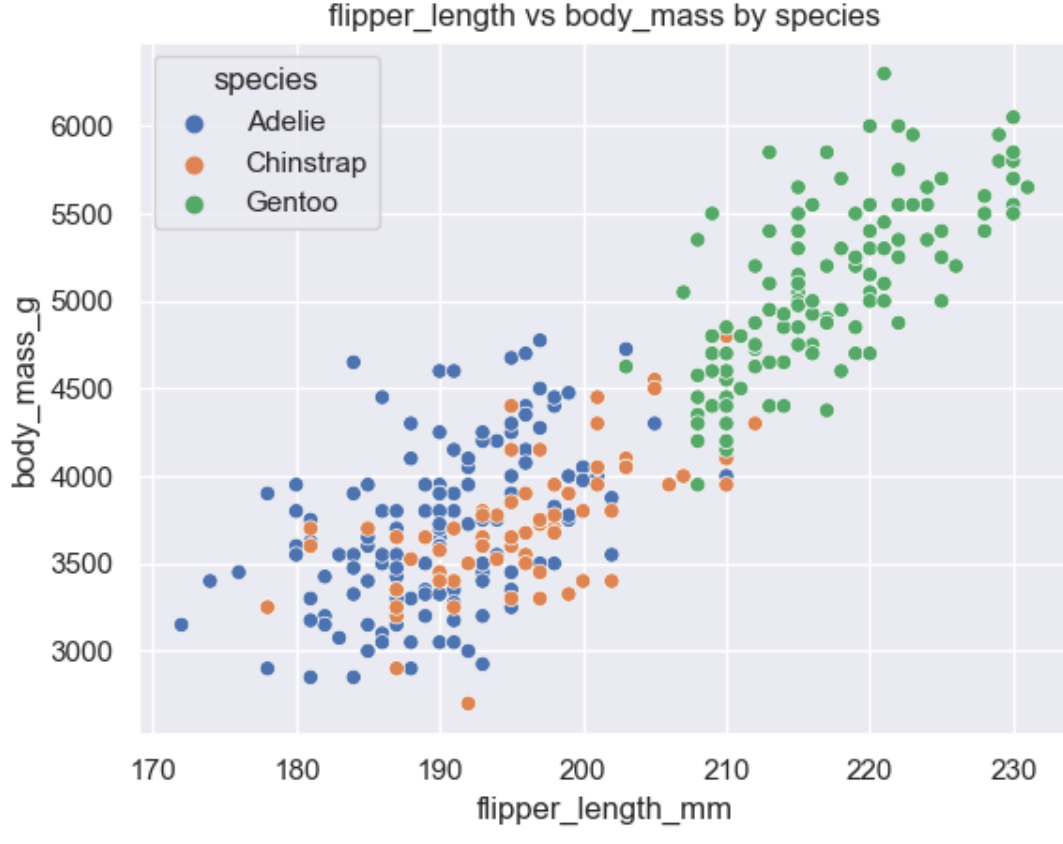


In [35]:

```
clf = SVC()
clf.fit(X_train, y_train);
training_accuracy = (clf.predict(X_train) == y_train).mean()
testing_accuracy = (clf.predict(X_test) == y_test).mean()
```

Classifiers

Classification is one of the major tasks in machine learning. Its goal (as its name implies) is to ingest data associated with a label and be able to predict the label of future unseen data by learning some sort of pattern. Within our penguins dataset, this would involve predicting the penguin species ('species') based on its 'flipper_length_mm' and 'body_mass_g'.



In [35]:

```
clf = SVC()
clf.fit(X_train, y_train);
training_accuracy = (clf.predict(X_train) == y_train).mean()
testing_accuracy = (clf.predict(X_test) == y_test).mean()
```

training accuracy: 0.745

testing accuracy: 0.6940298507462687

Why are we talking about this?

Foreshadowing! You may (or may not) need to apply some of this knowledge (relatively) soon! Possibly for your project

K Nearest Neighbors (KNN) for Classification

idea: Given an unseen value of flipper_length and body_mass, how can we determine what species of penguins it belongs to?

The KNN approach is simple - if I look at the k nearest points to this new value, and I take the most common species among them, then this point is **most likely** the same species as the most common species among the neighbors.


flipper_length vs body_mass by species
species
Adelie
Chinstrap
Gentoo
6000
5500
5000
4500
4000
3500
3000
body_mass_g
170
180
190
200
210
220
230
flipper_length_mm

Procedure

Step 1: Given a new point *X*, quantify the distance between all points and *X*

Step 2: Take the **k-nearest** points to *X* into consideration

Step 3: Classify *X* as the most common label among the neighbors

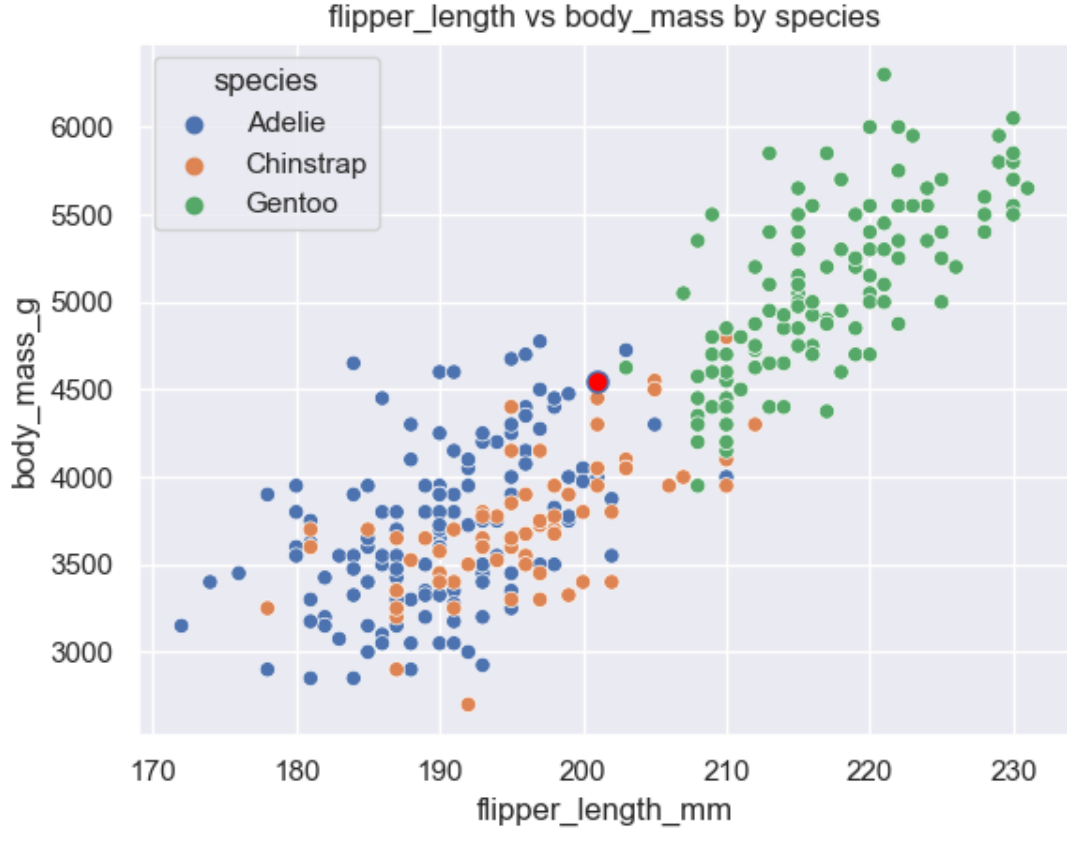
note: As the name implies, the key parts of this algorithm are **k** and **nearest**.

k is often referred to as a **hyper-parameter**, or a value that you choose independently. There are often procedures to select a good k, but this varies depending on the context.

Nearest refers to quantifying distance - one such way is to use euclidian distance (distance formula), but there are many other "distances" that can be used (ex. Manhattan Distance).

Checkpoint

Given the new data point (201, 4750), what would a KNN classifier classify the new point as for **k=1**? What about for **k=4**? What about for **k=10**?



Checkpoint Solution

for k=1: Seems like the orange point is closest according to an eye test, so Chinstrap

for k=4: The 3 closest points seem to all be from unique colors, the fourth point is hard to determine by eye. Could easily be Adelie or Chinstrap

for k=10: The bulk of the points nearby are blue, so Adelie

Thanks for coming!