Introduction to programming with Python

By Lukas Jarosch & Leonhard Kohleick

Variables and basic data types

Our very first program

```
# Welcome to the Seminar
message = "Hello all!"

print(message)
```

```
Hello all!
```

Our very first program

Variable assignment

```
a = 1
b = a

# you can also print two values at the same time
print(a, b)
```

```
1 1
```

Variable assignment

```
a = 1
b = a
a = 2

print(a, b)
```

```
2 1
```

`print()` with different separators

```
a = 1
b = 2
print(a, b)

# different separators
print(a, b, sep="--")
print(a, b, sep="\n")
```

```
1 2
1--2
1
2
```

special “newline” character

Variable names

Allowed: letters, underscores (“_”), numbers (but not at the start of the variable name)

Examples:

```
myvariable

MyVariable

my_variable

my_variable_1
```

Variable names

Not allowed: special characters, numbers at the start

Examples:

```
my-variable

myvariable!

1_variable
```

Basic Data Types

```
## the three basic types
x = 1
y = 1.4
z = "hello"
h = 'hello'

print(type(x), type(y), type(z), type(h))
```

integer (= whole number)

float (= decimal number)

string (= character sequence)

```
<class 'int'> <class 'float'> <class 'str'> <class 'str'>
```

Conversion between Data Types

```
## examples for type-conversion
x = int(2.8)
y = float(1)
z = str(2.5) # <-- equivalent to z = "2.5"

print(x, y, z)
print(type(x), type(y), type(z))
```

```
2 1.0 2.5
<class 'int'> <class 'float'> <class 'str'>
```

Basic math

```
# addition
print(5+5)

# multiplication
print(3*2)

# division
print(6/2)

# exponentiation
print(2**3)
```

```
10
6
3.0
8
```

Math with strings?

```
z = 2
print(2*z)
print(2*str(z))
```

?

Math with strings?

```
z = 2
print(2*z)
print(2*str(z))
```

```
4
22
```

Math with strings?

```
# "math" with strings
x = "hello "
y = "world"
print(x+y)
print(5*x)
print(2*x + y)
```

```
hello world
hello hello hello hello hello
hello hello world
```

Math with strings?

```
print("hello" + 10) # <-- this does not work
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_25352/3777394797.py in <module>
     11
     12 print("hello"+"10")
---> 13 print("hello" + 10) # <-- this does not work

TypeError: can only concatenate str (not "int") to str
```

Our first more complex program

```
friend = "Peter"
birthyear = 1998

message = "Hi " + friend + ", congratulations to becoming " + str(2021 - birthyear) + " years old!!"
print(message)
```

```
Hi Peter, congratulations to becoming 23 years old!!
```

Format-strings: a very handy utility

```
friend = "Peter"
birthyear = 1998

# with a regular string
message = "Hi " + friend + ", congratulations to becoming " + str(2021 - birthyear) + " years old!!"
print(message)
```

```
# with a format-string
message = f"Hi {friend}, congratulations to becoming {2021 - birthyear} years old!!"
print(message)
```

Variables and basic data types
Exercises

Exercise 0: Bonus

As a tradition among programmers, your first script should be dedicated to greeting the world.
Print the words "Hello World!".

In [ ]:

Exercise 1

Assign two variables to the values 3 and "A" and print their values. What data types do these variables belong to?

In [ ]:

Exercise 2

The following code seems to have a few mistakes in it. Can you fix them without assigning variables to new values?

In [1]:
```
1var = 5
2var = "10"

# desired output is 15
print(1var + 2var)
```

```
  File "C:\Users\jarosch\AppData\Local\Temp/ipykernel_18240/1395479450.py", lin
e 1
    1var = 5
     ^
SyntaxError: invalid syntax
```

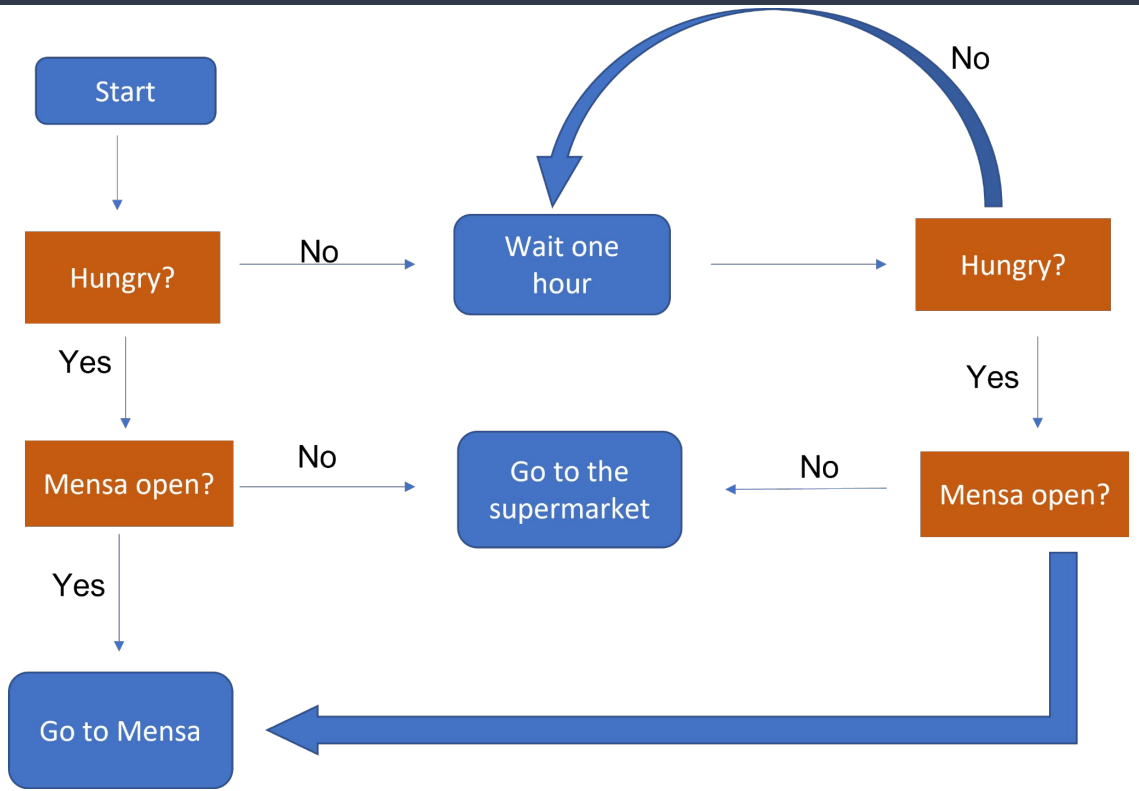
Exercise 3

Can you print the sequence "103103103" from the following variables, without explicitly using/assigning any new values? Remember the differences between addition/multiplication with strings and numbers.

In [4]:
```
x = 8
y = 2
z = 3
```

Boolean Operators and Conditions

Flow Control: After your Biochemistry Lecture



Boolean Operators

```
1 print(True and False)
2 print(True or False)
3 print(not True)
```

```
False
True
False
```

Comparison Operators

`==` : Are they the same ? (do not use `=` because this means assignment)

`!=` : Are they not the same?

```
# Compare the number of genes

human = 21000 #homo sapiens
rice = 38000  #oryza sativa

print(human == rice)
print(human != rice)
```

```
False
True
```

Less/Greater than

`>` : Greater than

`<` : Less than

```
print(rice > human)
print(rice > rice)
print(human > rice)
print(human < rice)
```

```
True
False
False
True
```

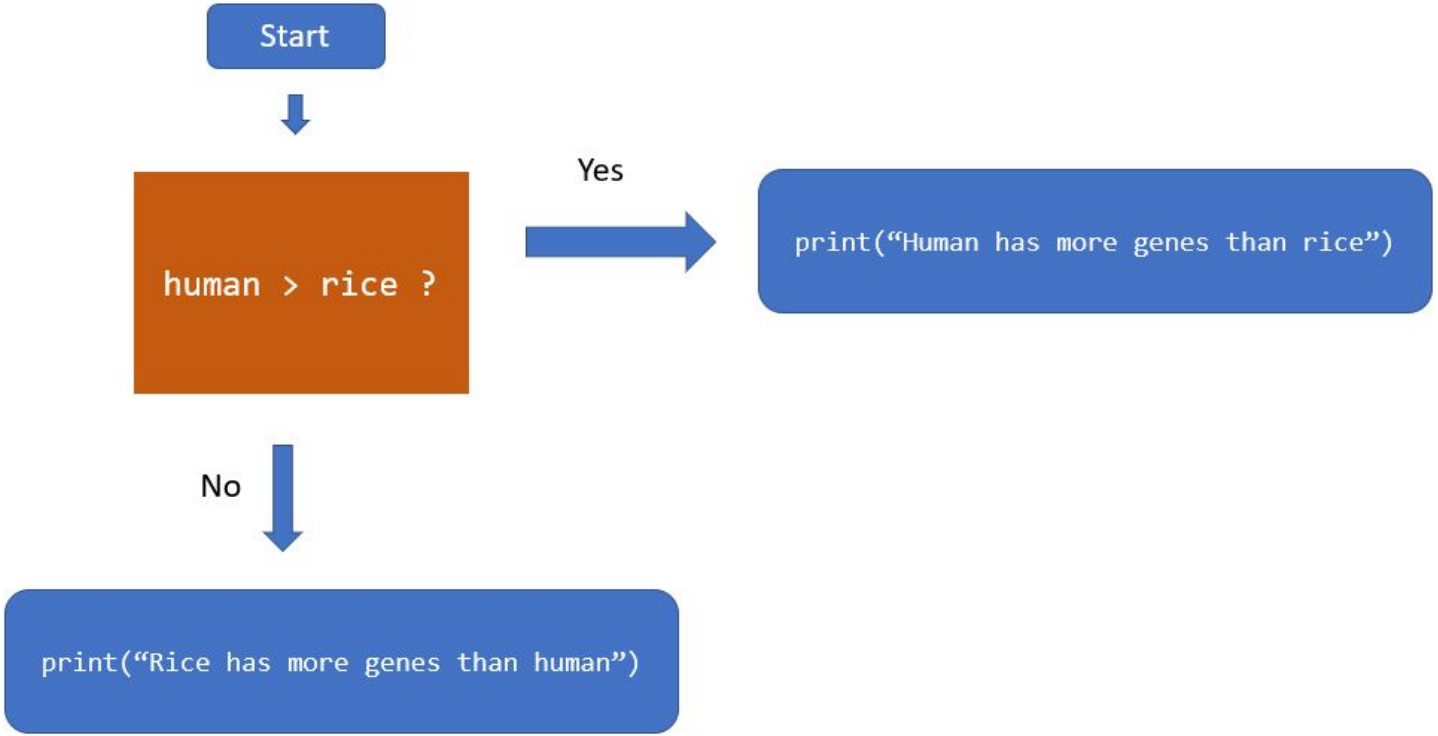
Less/Greater than or equal to

`<=` : Less than or equal to

`>=` : Greater than or equal to

```
print(rice >= human)  True
print(human >= human) True
print(human <= rice)  True
```

Conditions


Start
human > rice ?
Yes
print("Human has more genes than rice")
No
print("Rice has more genes than human")

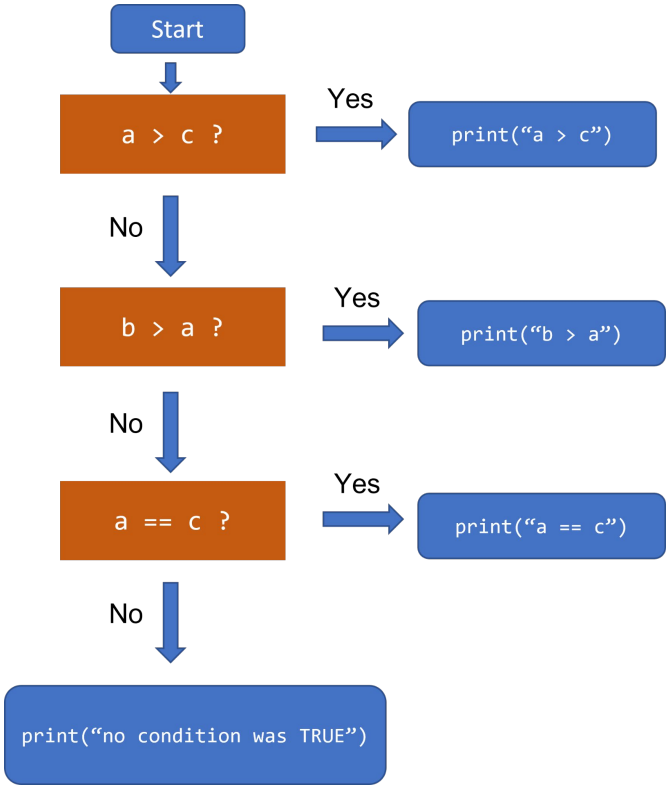
if statements are crucial in programming

```
1  #Compare the number of genes
2  if human > rice:
3      print("Human has more genes than rice")
4  else:
5      print("Rice has more genes than human")
```

```
Rice has more genes than human
```

Complex flow controls:

```
1  a = 5
2  b = 4
3  c = 7
```



Using the full `if .. else` statement is chaotic

```
1  a = 5
2  b = 4
3  c = 7
4
5  if a > c:
6      print("a > c")
7  else:
8      if b > a:
9          print("b > a")
10     else:
11         if a == c:
12             print("a == c")
13         else:
14             print("No condition was TRUE")
15
16
```

```
No condition was TRUE
```

`elif` (`else if`) avoids the chaos

```
1  a = 5
2  b = 4
3  c = 7
4
5  if a > c:
6      print("a > c")
7  elif b > a:
8      print("b > a")
9  elif a == c:
10     print("a == c")
11 else:
12     print("No condition was TRUE")
```

```
No condition was TRUE
```

Nested boolean conditions

```
1 a = 5
2 b = 4
3 c = 7
```

```
1 if ( a > b and c > a ):
2     print("Hooray")
3 else:
4     print("Sadness")
```

```
Hooray
```

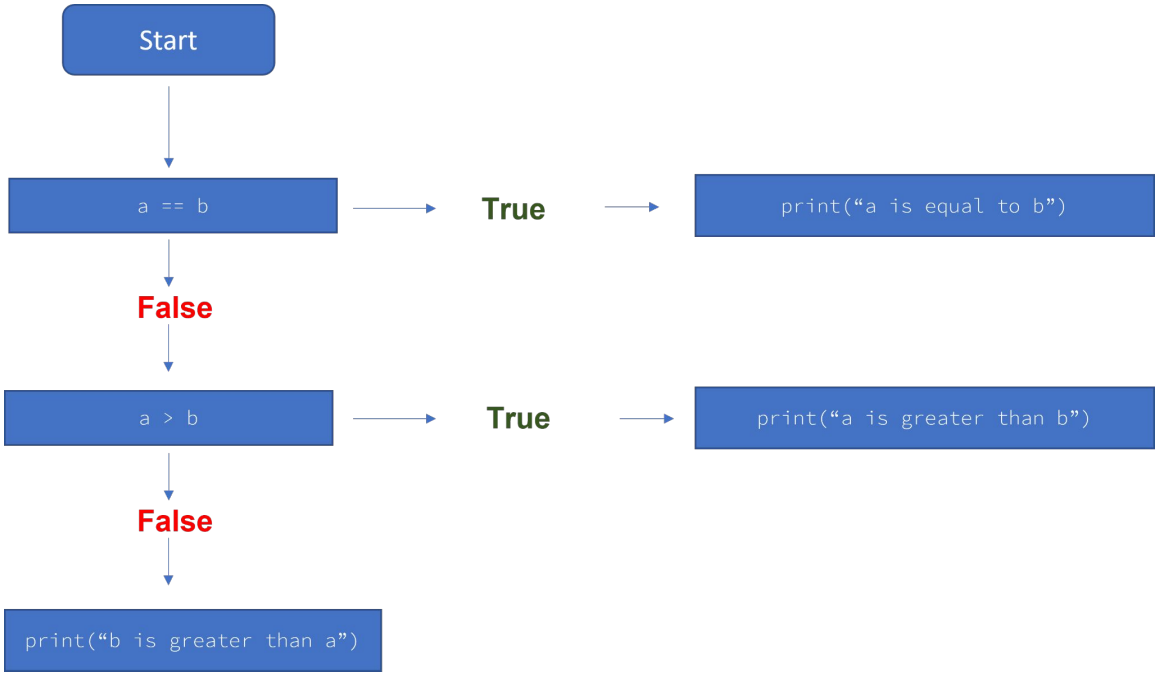
```
1 if ( a > b or c < a ) == False:
2     print("Hooray")
3 else:
4     print("Sadness")
```

```
Sadness
```

Boolean Operators and Conditions
Exercises

Exercise 1

Write a program that resembles this flowchart:



Exercise 2

Which value should we assign to `d` to set the condition to `True` ?

In [5]:
```
1 a = 5
2 b = 6
3 c = 7
4 d = "8"
5
6 ((a == (b - 1)) and c == d ) or (a + 1 == d)
```

Out[5]: False

Exercise 3

What sequences below are identical?

[6]:

```
1 sequence_1 = "MTQLQISLLLTATISLLHLVVATPYEAYPIGKQYPPVARVNESFTFQISNDTYKSSVDKTAQITYNCF
2 sequence_2 = "MTQLQISLLLTATISLLHLVVATPYEAYPIGKQYPPVARVNESFTFQISNDTYKSSVDKTAQITYNCF
3 sequence_3 = "MTQLQISLLLTATISLLHLVVATPYEAYPIGKQYPPVARVNESFTFQISNDTYKSSVDKTAQITYNCF
```

Lists and Tuples

Lists store multiple objects in one variable

```
mylist = [1, 2, 3, 4]
print(mylist)
```

```
[1, 2, 3, 4]
```

```
mylist = list((1, 2, 3, 4))
print(mylist)
```

```
[1, 2, 3, 4]
```

Lists can store any kind of data type

```
mylist = [True, 2.3, "Hello", [1, 2]]
```

even other lists!

String to list conversion

```
mysequence = "ATTCGCTA"
mylist = list(mysequence)
print(mylist)
```

```
['A', 'T', 'T', 'C', 'G', 'C', 'T', 'A']
```

List to string conversion

```
mysequence = str(mylist)
print(mysequence)
print(type(mysequence))
```

```
['A', 'T', 'T', 'C', 'G', 'C', 'T', 'A']
<class 'str'>
```

that is not really what we wanted...

List to string conversion with `.join()`

```
mysequence = "|".join(mylist)
print(mysequence)
```

```
A|T|T|C|G|C|T|A
```

custom separator

```
mysequence = "".join(mylist)
print(mysequence)
```

```
ATTCGCTA
```

Tuples

```
mytuple = tuple((True, 1, "ABC", [1, 2]))
print(mytuple)
```

```
(True, 1, 'ABC', [1, 2])
```

```
mytuple = (True, 1, "ABC", [1, 2])
print(mytuple)
```

```
(True, 1, 'ABC', [1, 2])
```

Tuples vs Lists

- Tuples are **immutable** while lists are **mutable**
- Tuples are more **memory efficient** and needed for specific operations
- For this course you do not really need to care about tuples but you should recognize them when you see them in Python code

Indexing: [start]

```
# faster: mylist = list("Biochemistry")
mylist = ['B','i','o','c','h','e','m','i','s','t','r','y']
print(mylist)
```

```
['B', 'i', 'o', 'c', 'h', 'e', 'm', 'i', 's', 't', 'r', 'y']
```

0	1	2	3	4	5	6	7	8	9	10	11
B	i	o	c	h	e	m	i	s	t	r	y

Indexing: [start]

```
print(mylist[0])
```

```
B
```

0	1	2	3	4	5	6	7	8	9	10	11
B	i	o	c	h	e	m	i	s	t	r	y

Indexing: [start]

```
print(mylist[1])
```

```
i
```

0	1	2	3	4	5	6	7	8	9	10	11
B	i	o	c	h	e	m	i	s	t	r	y

Indexing: [start]

```
print(mylist[-1])
```

```
y
```

-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6	7	8	9	10	11
B	i	o	c	h	e	m	i	s	t	r	y

Indexing: [start]

```
print(mylist[-4])
```

```
s
```

-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6	7	8	9	10	11
B	i	o	c	h	e	m	i	s	t	r	y

Indexing: [start:stop]

```
print(mylist[2:4])
```

```
['o', 'c']
```

note that stop itself is not included

-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6	7	8	9	10	11
B	i	o	c	h	e	m	i	s	t	r	y

Indexing: `[start:stop]`

```
print(mylist[:3])
```

```
['B', 'i', 'o']
```

not specifying `start` or `stop` includes all values from the left/right

-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6	7	8	9	10	11
B	i	o	c	h	e	m	i	s	t	r	y

Indexing: [start:stop]

```
print(mylist[3:])
```

```
['c', 'h', 'e', 'm', 'i', 's', 't', 'r', 'y']
```

not specifying `start` or `stop` includes all values from the left/right

-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6	7	8	9	10	11
B	i	o	c	h	e	m	i	s	t	r	y

Indexing: [start:stop:step]

```
print(mylist[3:9:2])
```

```
['c', 'e', 'i']
```

-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6	7	8	9	10	11
B	i	o	c	h	e	m	i	s	t	r	y

Indexing also works with strings

```
mystring = "Bioinformatics"
print(mystring[5:-5])
```

```
form
```

Indexing: reassignment

```
mylist = [1, 2, 2, 4, 5, 7, 8]
print(mylist)

# assign a single element
mylist[2] = 3
print(mylist)

# assign multiple elements
mylist[-2:] = [6, 7]
print(mylist)
```

```
[1, 2, 2, 4, 5, 7, 8]
[1, 2, 3, 4, 5, 7, 8]
[1, 2, 3, 4, 5, 6, 7]
```

“Math” with lists

```
mylist = [1, 2, 3]
print(mylist + ["A", "B"])
print(mylist * 2)
```

```
[1, 2, 3, 'A', 'B']
[1, 2, 3, 1, 2, 3]
```

List functions and methods

Adding/removing values

```
mylist = ["A", "B", "C", "D"]
```

Syntax	Output	Description
mylist.append("E")	["A", "B", "C", "D", "E"]	add an element to the end of the list
mylist.extend(["E", "F"])	["A", "B", "C", "D", "E", "F"]	add the elements of another list to the list
mylist.insert(1, "a")	["A", "a", "B", "C", "D", "E"]	insert an element at a specific index
mylist.pop(2)	["A", "B", "D", "E"]	remove an element at a specific index
mylist.remove("B")	["A", "C", "D", "E"]	remove the first occurence of an item

List functions and methods

Finding elements

```
mylist = ["A", "B", "B", "C", "D"]
```

Syntax	Output	Description
`mylist.index("B")`	1	returns the index of the item's first occurence
`mylist.count("B")`	2	returns the number of times an item occurs in the list
`"D" in mylist`	True	returns True if the item is in the list, False if it is not

List functions and methods

Other useful functions

```
mylist = ["A", "B", "B", "C", "D"]
```

Syntax	Output	Description
`mylist_2 = mylist.copy()`	`["A", "B", "B", "C", "D"]`	makes a copy of a list
`len(mylist)`	`5`	returns the length of the list

Lists and Tuples
Exercises

Exercise 1

For this exercise you are given the first 70 nucleotides of the Sars-Cov2 spike protein sequence.

```
spike = "ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAAT"
```

Exercise 1

Perform the following tasks on the sequence:

- print the first codon (= first three bases)
- create a new sequence without codons 4, 5, and 6

```

```

Exercise 2

Now, return a new sequence with the following mutations using Python functions:

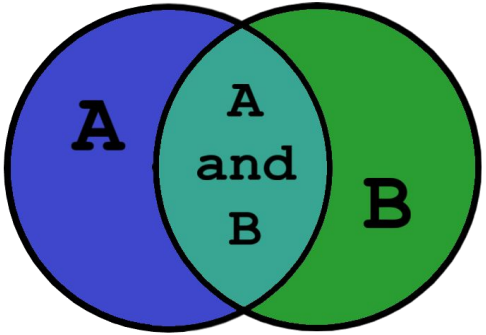
- T to G point mutation at position 6
- change the second last codon to "CGG"
- insert a C between nucleotide 1 and 2 of the sequence
- append a polyA-repeat of 10 nucleotides to the end of the sequence

Exercise 3 - Bonus

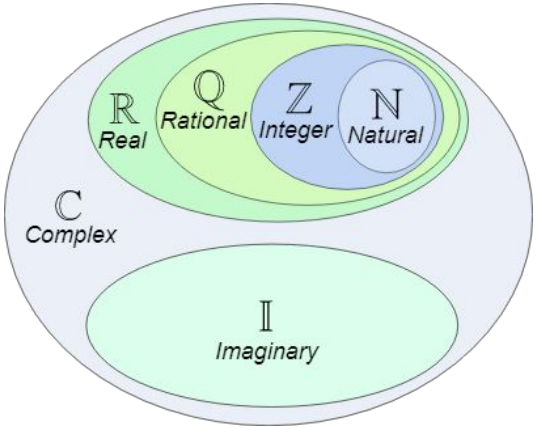
Let's pretend that the sequence "ATTGCC" is a binding motif for a specific transcription factor. Write a conditional clause that prints "sequence contains motif" if the motif is in the sequence and "sequence does not contain motif" otherwise and apply it to the spike sequence.

Sets and Dictionaries

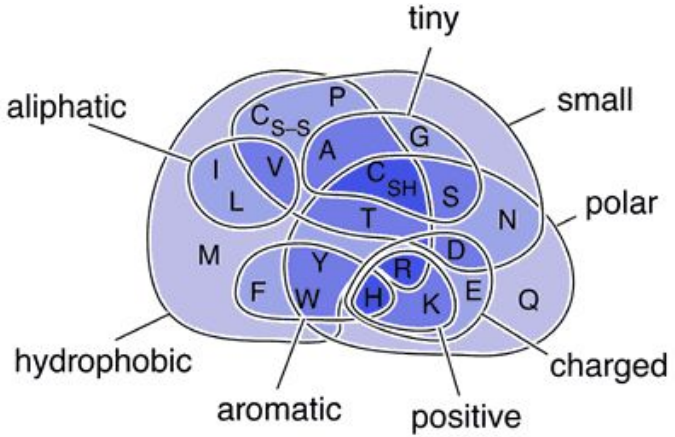
Sets



https://realpython.com/python-sets/



https://www.mathsisfun.com/sets/images/number-sets.svg



https://upload.wikimedia.org/wikipedia/commons/b/b5/Amino_Acids_Venn_Diagram.png

Sets in Python

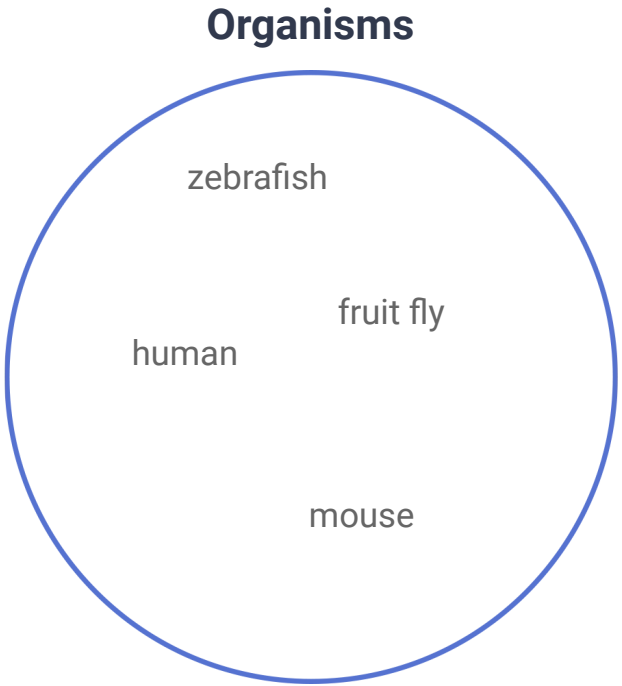
- Sets are **unordered** (no indexing)
- Sets contain only **unique** elements
- Sets can only contain **immutable** data types (e.g. tuples but no lists)

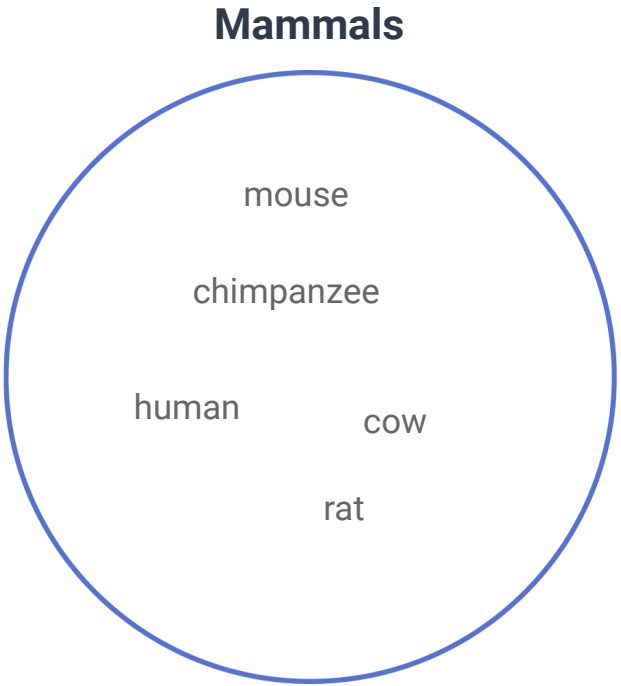
Sets in Python

```
# set definition
set_1 = {1, 3, 3, 2}
set_2 = set(("A", "A", True, 3.4, 2, 3, 1, (1, 5)))

# sets do not keep order information and automatically remove duplicates
print(set_1, set_2, sep = "\n")

# sets can be converted back to lists/tuples
print("\n")
print(list(set_1))
```

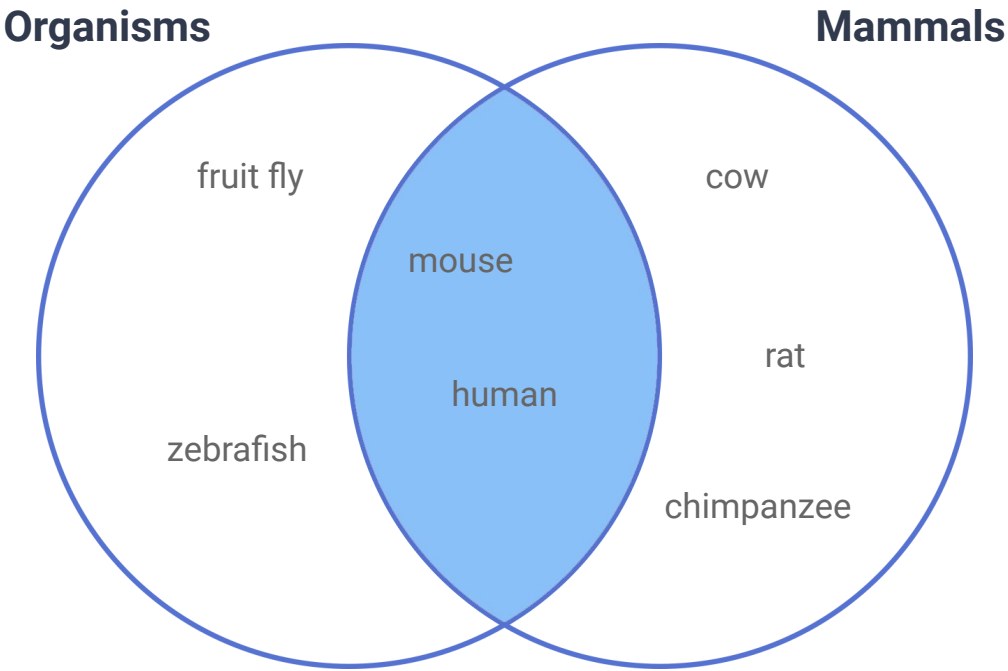
```
{1, 2, 3}
{True, 2, 3.4, 3, (1, 5), 'A'}


[1, 2, 3]
```

Set overlaps

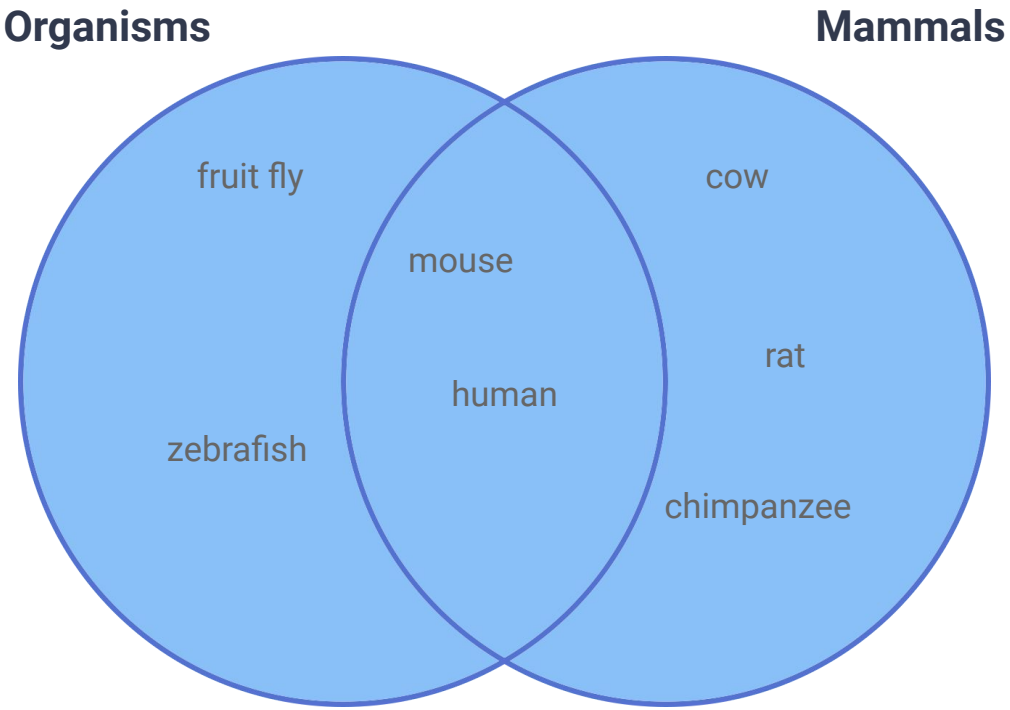




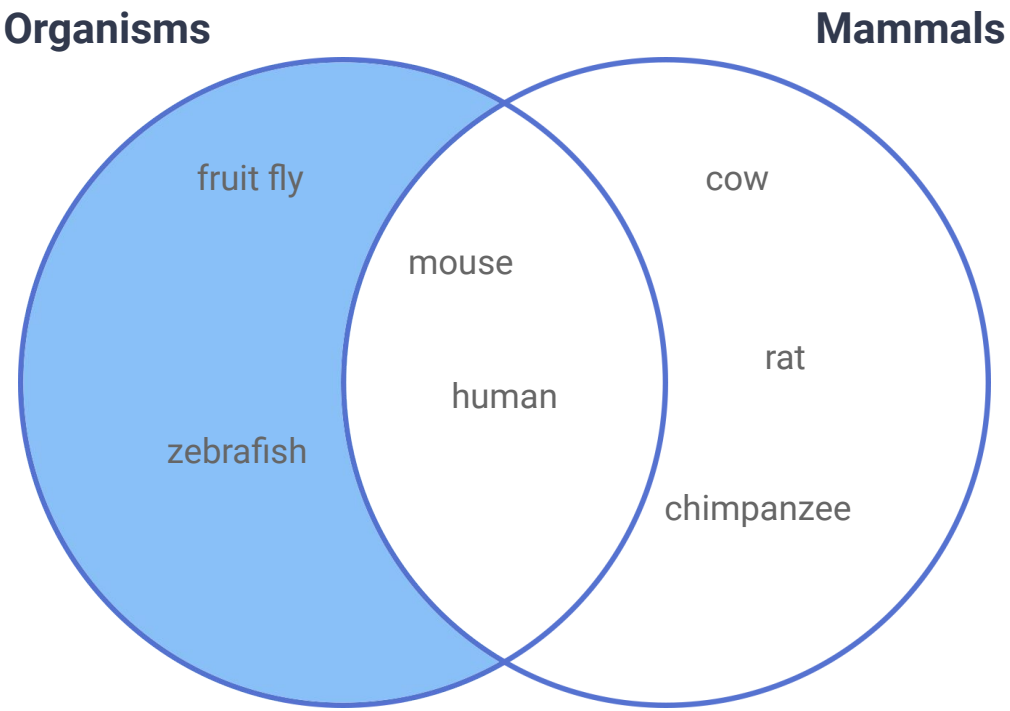
Set overlaps: Intersection



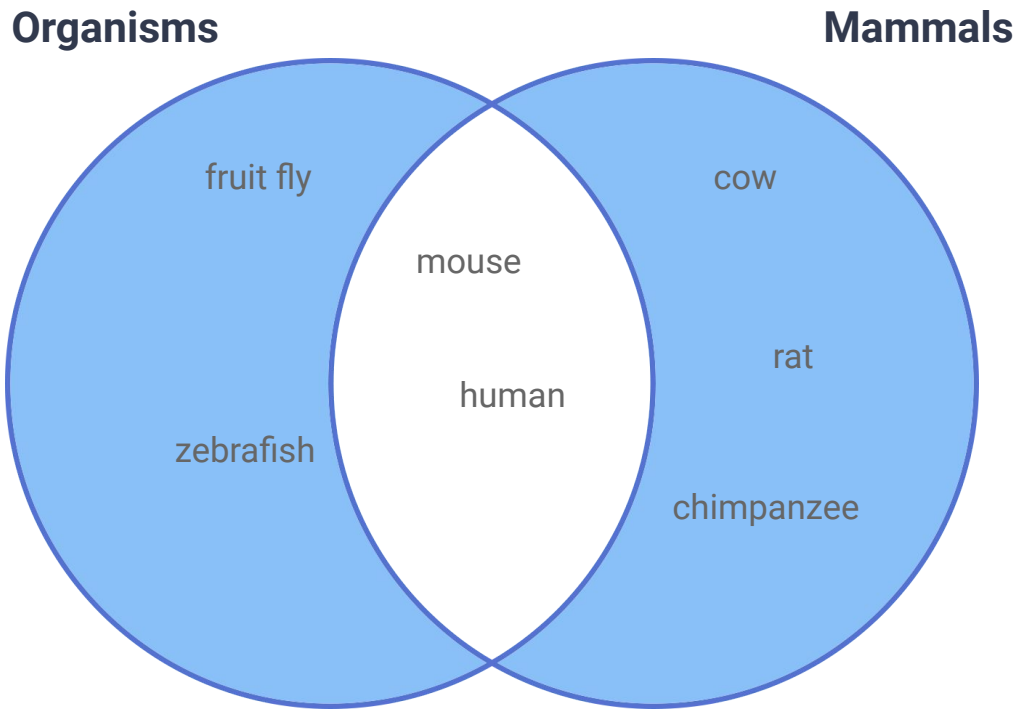
Set overlaps: Union



Set overlaps: Left-sided difference



Set overlaps: Symmetric difference



Set methods and functions

```
set_1 = {“A”, “B”, “C”}
set_2 =      {“B”, “C”, “D}
```

Syntax	Output	Description
set_1.difference(set_2)	{“A”}	return left-sided difference
set_1.symmetric_difference(set_2)	{“A”, “D”}	return symmetric difference
set_1.intersection(set_2)	{“B”, “C”}	return intersection
set_1.union(set_2)	{“A”, “B”, “C”, “D”}	return union

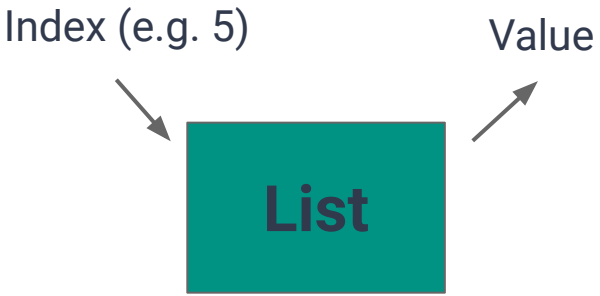
Set methods and functions

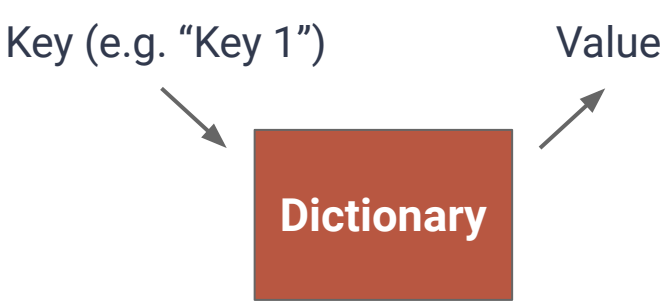
```
set_1 = {“A”, “B”, “C”}
set_2 =       {“B”, “C”, “D}
```

Syntax	Output	Description
set_1.add(“D”)	{“A”, “B”, “C”, “D”}	add an element to the set
set_3 = set_1.copy()	{“A”, “B”, “C”}	copy the set
len(set_1)	3	return length of the set
“C” in set_1	True	returns True if the item is in the set, False if it is not

Dictionaries

Dictionaries





Dictionaries in Python

```
# {} syntax
peter = {"Name": "Peter", "Age": 23, "Height": 1.91}

# dict() function
anna = dict([["Name", "Anna"], ["Age", 21], ["Height", 1.72]])

# dict() + zip()
keys = ["Name", "Age", "Height"]
values = ["Tim", 22, 1.74]
tim = dict(zip(keys, values))

print(peter, anna, tim, sep="\n")
```

```
{'Name': 'Peter', 'Age': 23, 'Height': 1.91}
{'Name': 'Anna', 'Age': 21, 'Height': 1.72}
{'Name': 'Tim', 'Age': 22, 'Height': 1.74}
```

You can organize data in dictionaries

Measurement 1	Measurement 2	Measurement 3
1.2	0.9	1.0
1.9	2.2	1.8
3.0	3.1	3.0
4.2	4.3	3.9

```
data = {
    "Measurement 1": [1.2, 1.9, 3.0, 4.2],
    "Measurement 2": [0.9, 2.2, 3.1, 4.3],
    "Measurement 3": [1.0, 1.8, 3.0, 3.9]
}
```

Indexing dictionaries

```
tim = {'Name': 'Tim', 'Age': 22, 'Height': 1.74}

# print out a value
print(tim["Name"])

# reassign a value
tim["Age"] = 25

# assign a new key-value pair
tim["Eye color"] = "blue"

print(tim)
```

```
Tim
{'Name': 'Tim', 'Age': 25, 'Height': 1.74, 'Eye color': 'blue'}
```

Dictionary methods and functions

```
dict_1 = {“A”: 1, “B”: 2}
dict_2 = {“C”: 3, “D”: 4}
```

Syntax	Output	Description
dict_1.update(dict_2)	{'A': 1, 'B': 2, 'C': 3, 'D': 4}	combine two dictionaries
dict_3 = dict_1.copy()	{'A': 1, 'B': 2}	copy the dictionary
dict_1.keys()	dict_keys([‘A’, ‘B’])	return dictionary keys (object can be converted into a list)
dict_1.values()	dict_values([1, 2])	return dictionary values (object can be converted into a list)

Bonus: the `defaultdict()` method

```
from collections import defaultdict

normal_dict = {}
default_dict = defaultdict(list)

# you need to create the list first before you can append to it
normal_dict["new_key"] = []
normal_dict["new_key"].append(1)

# defaultdict will automatically insert an empty list at each new key
print(default_dict["new_key"])
default_dict["new_key"].append(1)

print(normal_dict, default_dict, sep="\n")
```

```
[]
{'new_key': [1]}
defaultdict(<class 'list'>, {'new_key': [1]})
```

you can find a more detailed explanation in the Jupyter Notebook!

Sets and Dictionaries
Exercises

Exercise 1

Below you can find two amino acid peptide sequences as well as a set of all 20 amino acids. Use python sets to determine which unique amino acids occur in both sequences and which amino acids occur in neither of the two sequences.

```
peptide_1 = "NRLTPQPMPQ"
peptide_2 = "LTGANQVDRG"

amino_acids = {'A','R','N','D','C','E','Q','G','H','I','L','K','M','F','P','S','T','W','Y','V'}
```

Exercise 2 - Part 1

You did an experiment which gave you the following data:

Measurement 1	Measurement 2	Measurement 3	Measurement 4
1.3	0.9	1.0	1.1
2.9	1.7	2.3	2.3
3.2	2.8	2.9	3.0
4.0	3.7	4.2	3.7

Half of the data is already stored in a dictionary below. Add the remaining measurements to that same dictionary.

```
mydata = {
    "Measurement 1": [1.3, 2.9, 3.2, 4.0],
    "Measurement 2": [0.9, 1.7, 2.8, 3.7]
}
```

Exercise 2 - Part 2

Oops, while you check your data again you realise that the second value of measurement 1 should actually be 1.9 instead of 2.9!
Correctly reassign that value in `mydata`.

```
```

Exercise 3 - Bonus

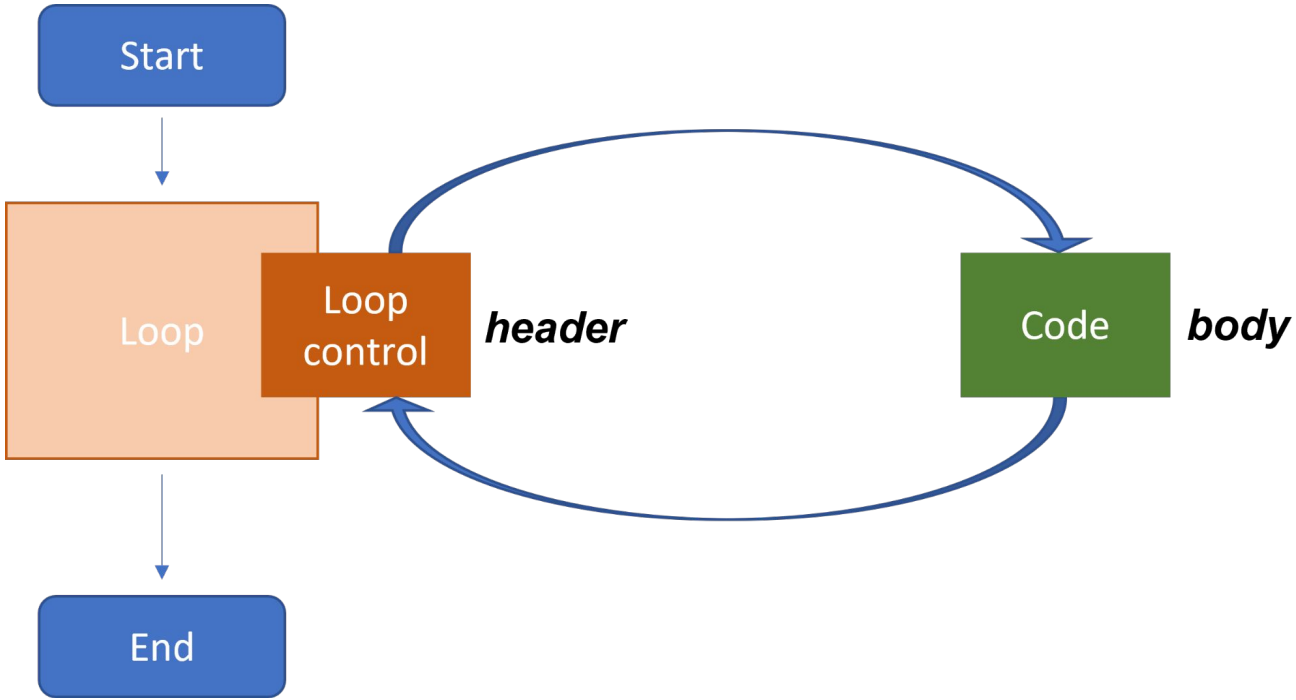
Below you are given a dictionary that translates common organism names into their scientific latin names and a variable called `common_name` that is assigned to the value "Human" as an example. Write a construct that prints the scientific counterpart of `common_name` but returns "Organism name not found!" when `common_name` is not in the dictionary's keys.

Can you also create a second dictionary that does the reverse translation from scientific names to common names?

```
common_to_scientific = {
    "Brewer's yeast": "Saccharomyces cerevisiae",
    "Thale cress": "Arabidopsis thaliana",
    "Nematode": "Caenorhabditis elegans",
    "Fruit fly": "Drosophila melangaster",
    "Zebrafish": "Danio rerio",
    "House mouse": "Mus musculus",
    "Human": "Homo sapiens",
}

common_name = "Human"
```

Loops

Loops: a sequence of instructions repeated until a certain condition is reached



`for` loops iterate over a defined instance

In [1]:

```
1 for amino_acid in "TEWQIPFV":          # this is the header
2     print(amino_acid)                  # this is the body
```

```
T
E
W
Q
I
P
F
V
```

The `range()` function returns a list of number

In [5]:

```
1 for i in range(10):
2     print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

You also can print a statement again, and again.....

In [3]:

```
1 for i in range(10):
2     print("Hello World!")
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

`range()` shares the `(start, stop, step)` syntax

In [4]:

```
1 print(list(range(5,10)))
2 print(list(range(0,10,3)))
3 print(list(range(-10,-100,-30)))
```

```
[5, 6, 7, 8, 9]
[0, 3, 6, 9]
[-10, -40, -70]
```

Thymine counter (`for` loop)

```
1  genetic_sequence = 'TTAATCTTGTGATACGATATGAGA'
2  T_count = 0
3
4  for nucleotide in genetic_sequence:
5      if nucleotide == 'T':
6          T_count += 1
7
8  print(T_count)
```

➡ 9

`while` loops can iterate indefinitely

In [6]:

```
1  number = 0
2
3  while number < 10:              # header
4      print(number)               # body
5      number = number + 1         # body
```

```
0
1
2
3
4
5
6
7
8
9
```

The Thymine counter (`while` loop)

```
1  T_count_while = 0
2  i = 0
3
4  while i < len(genetic_sequence):
5      if genetic_sequence[i] == "T":
6          T_count_while += 1
7      i += 1
8
9  print(T_count_while)
10 #Are the T_counts equal for both loops?
11 print(T_count_while, T_count)
```

```
9
9 9
```

Equal ? ➡

The `break` statement stops loop execution

In [15]:

```
1  import time
2
3  i = 0
4
5  while True:
6      i += 1
7      print(i)
8      time.sleep(0.5)
9      if i == 21:
10         break
```

We can fill up lists with a loop easily

In [22]:

```
1 int_list = []
2 for i in range(10):
3     int_list.append(i)
4 print(int_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Combining two loops allows even more versatility

```
1  list_of_primers = ["CGCAAATGGGCGGTAGGCGTG","GACTATCATATGCTTACCGT","CAGGAAACAGCTATGAC"]
2  list_of_t_counts = []
3
4  for sequence in list_of_primers:
5      t_count = 0
6      for nucleotide in sequence:
7          if nucleotide == 'T':
8              t_count += 1
9      list_of_t_counts.append(t_count)
10
11 print(list_of_t_counts)
```

2nd loop

1st loop

```
[3, 7, 2]
```

The `zip()` function allows you to combine two lists

```
1 list(zip(list_of_primers, list_of_t_counts))
```

```
[('CGCAAATGGGCGGTAGGCGTG', 3),
 ('GACTATCATATGCTTACCGT', 7),
 ('CAGGAAACAGCTATGAC', 2)]
```

Or to iterate over two lists in parallel

```
1  codons = ['GCA','AGA','GAT','AAT','TGT']
2
3  amino_acid = ['A','R','D','N','C']
4
5  for codon, aa in zip(codons,amino_acid):
6      print(codon,aa)
```

```
GCA A
AGA R
GAT D
AAT N
TGT C
```

Loops
Exercises

Exercise 1

The DNA Sequence below needs to be transcribed to RNA. Write a program with a loop that does this job!

```
1 dna = "CCACCCTCGTGGTATGGCTAGGCATTCAGGAACCGGAGAACGCTTCAGACCAGCCCGGACTGGGAACCT
2 dna_to_rna = {'A': 'A', 'C': 'C', 'G': 'G', 'T': 'U'}
3
```

Exercise 2

Compute the G & C content of the DNA sequence above with a Python program! (use a loop)
Is it over 50%?

Functions & Modules/Packages

Modules allow you to

```
1  import random
2
3  print(random.randint(0,10))
```

```
7
```

```
1  print(random.choice(['A', 'C', 'G', 'T']))
```

```
A
```

A common notation

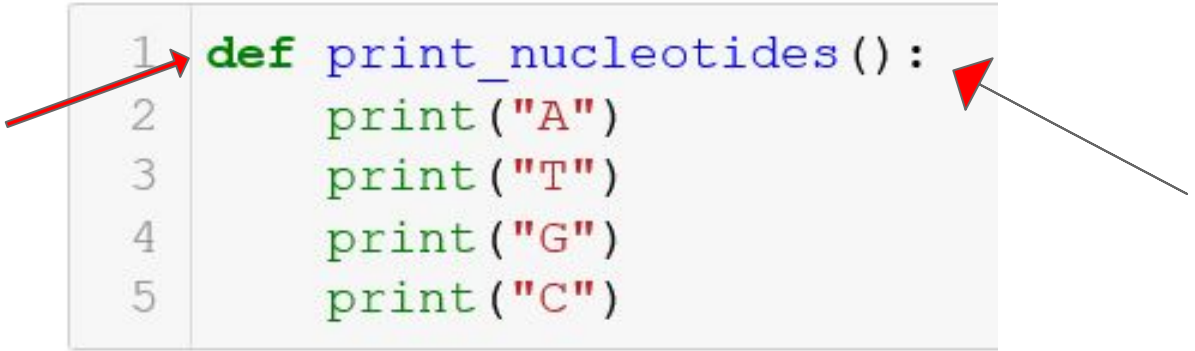
```
]:  1 import random as rd
    2 rd.randrange(10)
```

Loading a section of the package uses less memory

```
1 from random import randrange
2 randrange(10)
```

```
4
```

Functions are mini programs inside your code

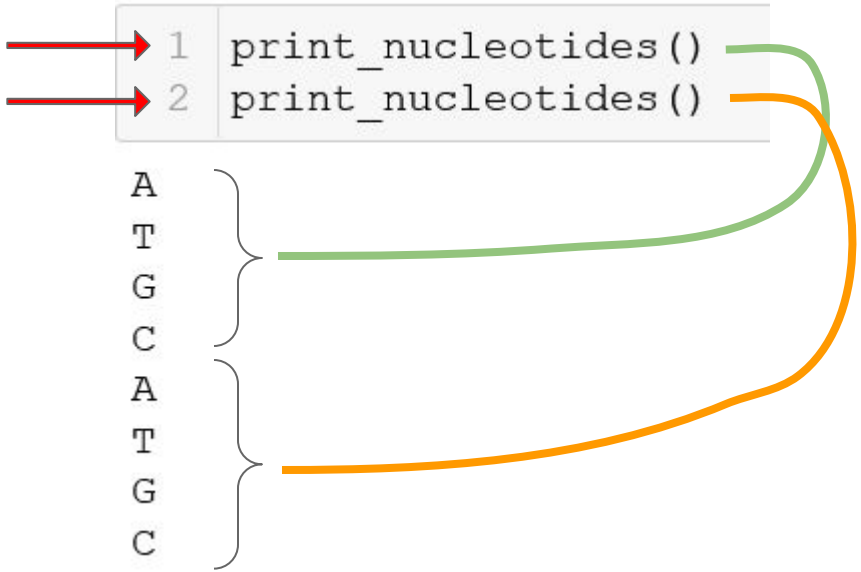


```
1  def print_nucleotides():
2      print("A")
3      print("T")
4      print("G")
5      print("C")
```

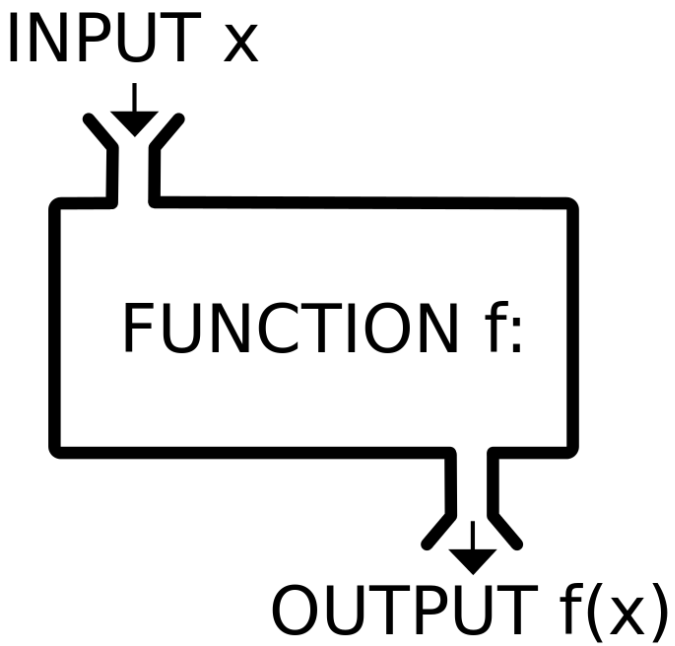
```
1  print_nucleotides()
```

```
A
T
G
C
```

You can call a functions as often as you want

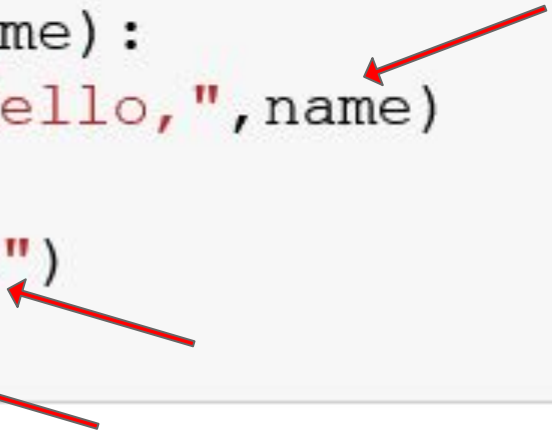


Function can receive a value and return a value



https://en.wikipedia.org/wiki/Function_(mathematics)#/media/File:Function_machine2.svg

```
1  def hello(name):
2      print("Hello,",name)
3
4  hello("Peter")
5  hello("Lea")
```



```
Hello, Peter
Hello, Lea
```

```
1  def square(a):
2      print(a**2)
3
4  square(4)
```

16

NoneType is returned per default

```
1  def square(a):
2      print(a**2)
3
4  b = square(4)
5
6  print(b + 9)
```

```
16

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-de0e98da6c50> in <module>
      4 b = square(4)
      5
----> 6 print(b + 9)

TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

Functions always return a value

```
1  def square(a):
2      return(a**2)
3
4  b = square(4)
5  print(b)
6  print(20*"-")
7  print(b + 9)
```

```
16
--------------------
25
```

Some more advanced functions:

```
1  def exponentiation(base,power):
2      return(base ** power)
3
4  print(exponentiation(2,3)) # 2**3
5  print(20*"-")
6  print(exponentiation(10,10))
7
```

```
8
--------------------
10000000000
```

Global and local variables differ

```
1  name = "Bob" # global variable
2
3  def myfunc():
4    name = "Alice" # local variable
5    print("Hello" , name)
6
7  myfunc()
8  print(20*"-")
9  print("Hello" , name) # name is Bob, although we changed the variable in the function
```

```
Hello Alice
--------------------
Hello Bob
```

Exercises 1 & 2

Exercise 1

Import the `math` package and print out the number π with it (Google is your friend!)

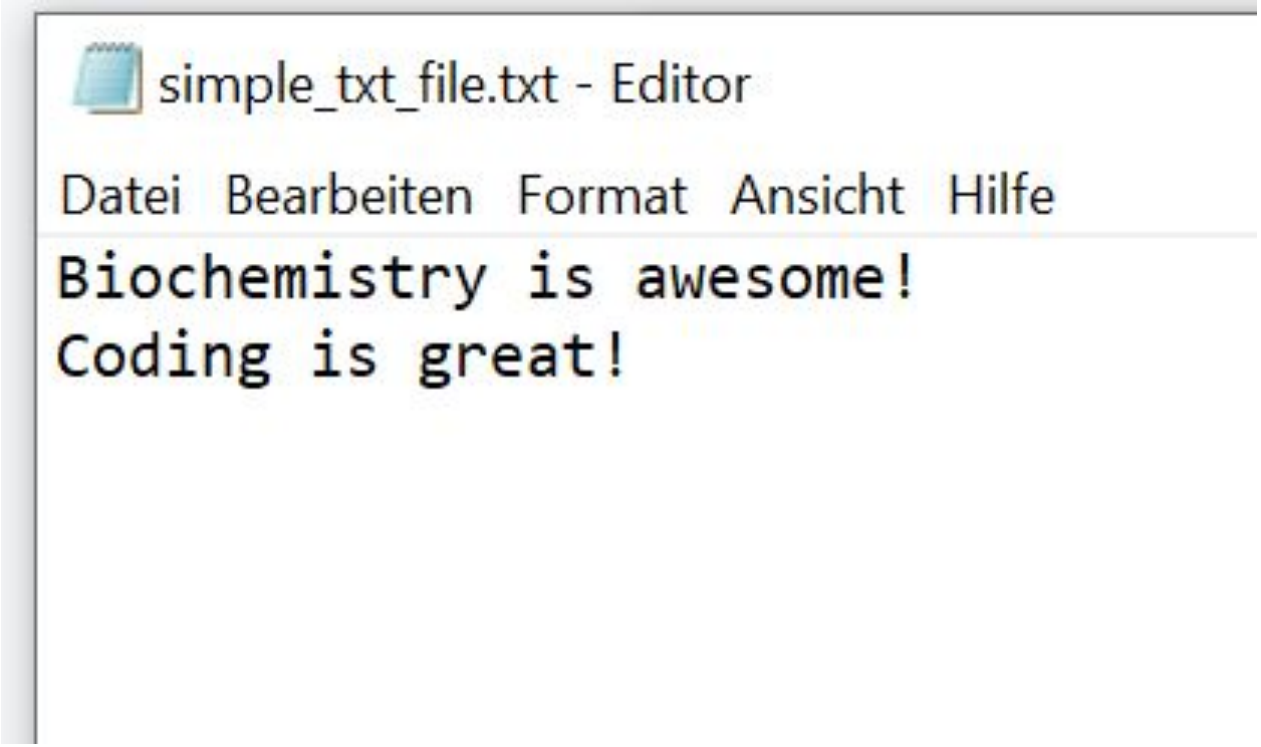
Exercise 2

Create a function that **returns** the number of point mutations between two DNA sequences. The two sequences below should be used as arguments:

```
[ ]:
1 sequence_1 = "GAGCCTACTAACGGGAT"
2 sequence_2 = "CATCGTAATGACGGCCT"
```

Working with CSV and TXT files

Here we have a very simple text file



Opening files with Python is easy

```
1 f = open("additional_data/simple_txt_file.txt",mode="r")
```

```
:  1 s = f.read()
   2 print(s)
```

```
Biochemistry is awesome!
Coding is great!
```

Reading line by line with `readline()`

```
1  f = open("additional_data/simple_txt_file.txt",mode="r")
2  line1 = f.readline()
3  line2 = f.readline()
4
5  print("Line 1:", line1)
6  print(20*"-")
7  print("Line 2:", line2 )
```

```
Line 1: Biochemistry is awesome!

--------------------
Line 2: Coding is great!
```

Closing files is crucial !

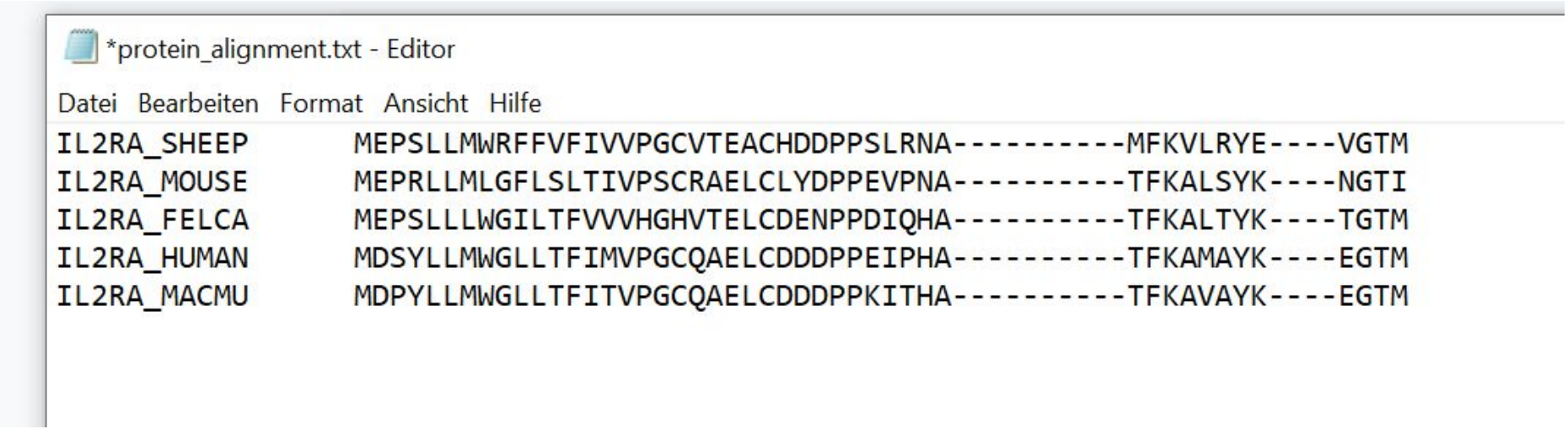
```
1 f.close()
```

`with open() as X:` closes the file automatically

```
1  with open("additional_data/simple_txt_file.txt",mode="r") as f:
2      line1 = f.readline()
3      line2 = f.readline()
4
5      print("Line 1:", line1)
6      print("Line 2:", line2)
```

```
Line 1: Biochemistry is awesome!

Line 2: Coding is great!
```

Now we have a sequence alignment in a specific format:



A parser for our sequence alignment

```
1 gene_seq = {}
2 with open("additional_data/protein_alignment.txt",mode="r") as f:
3     for line in f:
4         values = line.split() #splits the line at whitespace positions
5         gene_seq[values[0]] = values[1]
6 print(gene_seq)
```

```
{'IL2RA_SHEEP': 'MEPSLLMWRFFVFIVVPGCVTEACHDDPPSLRNA----------MFKVLRYE----VGTM',
```

Writing files in Python allows direct output of data

`mode = "w"`: will overwrite any existing content

```
1 with open('additional_data/new_txt_file.txt', mode='w') as t:
2     t.write('A new file has been created!')
```

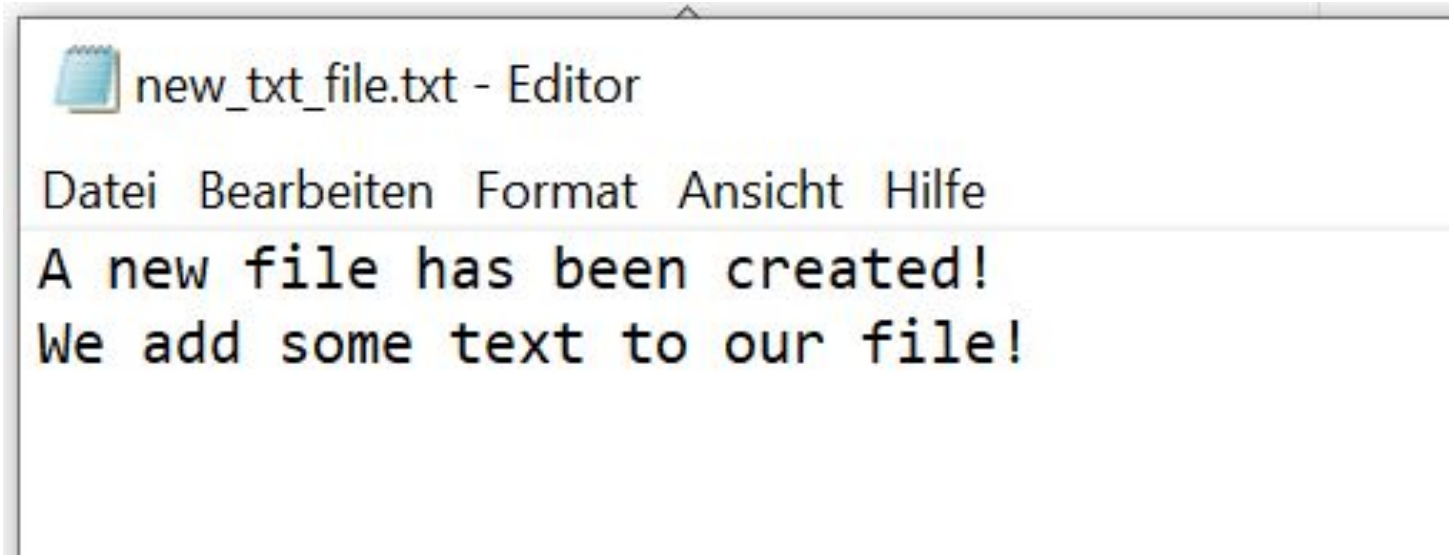
A newly written file:

The `mode()` is very crucial in writing files

`mode = "a":` will append to the end of the file

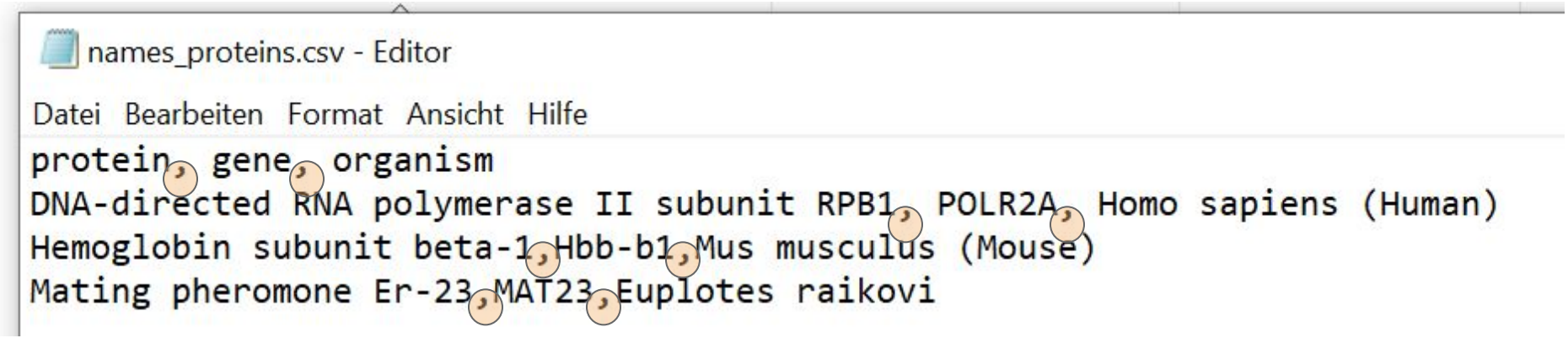
```
1 with open('additional_data/new_txt_file.txt', mode='a') as t:
2     t.write('\nWe add some text to our file!')
```

Appended text to file



CSV Files are crucial for data handling

Comma Separated Values



In a CSV viewer

	protein	gene	organism
	DNA-directed RNA polymerase II subunit RPB1	POLR2A	Homo sapiens (Human)
	Hemoglobin subunit beta-1	Hbb-b1	Mus musculus (Mouse)
	Mating pheromone Er-23	MAT23	Euplotes raikovi

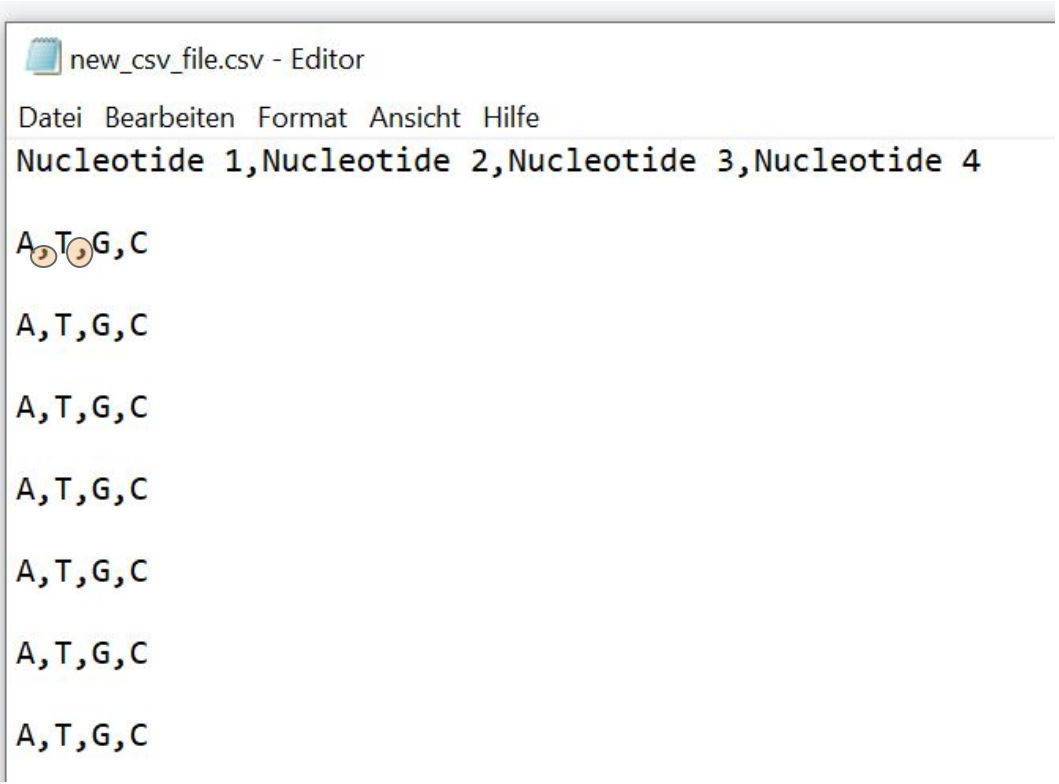
Reading CSV files with the csv parser:

```
import csv # the python csv parser

with open("additional_data/names_proteins.csv") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=",") # we initiate a reader object which removes the delimiter
    counter_line = 0
    header = []
    for row in csv_reader: # Looping though a csv_reader object, the row is now a list with elements
        if counter_line == 0:
            print('The categories in this csv file are:',row[0],row[1],row[2])
            header = row
            counter_line += 1
        else:
            print(f'{header[0]}:{row[0]}, {header[1]}:{row[1]}, {header[2]}:{row[2]}')
            counter_line += 1
```

You can also write CSV files

```
1  import csv
2
3  with open('additional_data/new_csv_file.csv', mode='w') as output_file:
4      output_file = csv.writer(output_file, delimiter=',') # we now create a writer objec
5      output_file.writerow(['Nucleotide 1','Nucleotide 2','Nucleotide 3','Nucleotide 4'])
6      for i in range(100):
7          output_file.writerow(['A','T','G','C'])
```

A fully functional CSV file is created:



Working with CSV and TXT files

Exercise

Exercise

In this one large exercise, we provide you with a file called `subcellular_location_data.csv`.

Your task is now to sort the gene names after their main cellular locations. Please provide each possible cellular location in the `csv` file with a list of gene names, in a dictionary. The main cellular location should be the key, the string of gene names should be the values.

Then, write out this dictionary as a new `csv` where the cellular location is in one column and all coresponding gene names in the other (in a large text string).

Pro Tip: use `from collections import defaultdict`

Introduction to Plotting

Python plotting libraries

Matplotlib

Pros

- is very customizable
- the go-to library for plotting in Python
- can do almost everything

Cons

- not very pretty default settings
- lots of “boilerplate” code

Matplotlib & Seaborn

Pros

- is very customizable
- the default library for plotting in Python
- can do almost everything

Cons

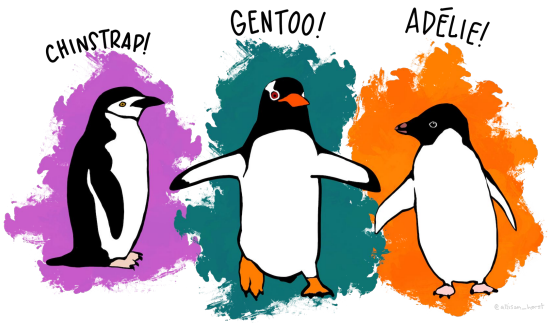
- not very pretty default settings
- lots of “boilerplate” code

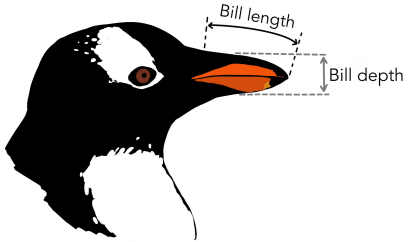
Pros

- provides many good-looking predefined plotting functions
- built with matplotlib, so you can still use matplotlib functions

Today's dataset: The palmer penguins

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
5	Adelie	Torgersen	39.3	20.6	190.0	3650.0	Male
...	...	...	...	...	...	...	...
338	Gentoo	Biscoe	47.2	13.7	214.0	4925.0	Female
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Female
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Female
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

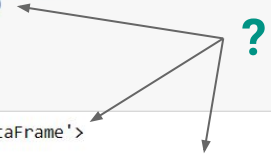




Artwork by @allison_horst

How to load the data

```
import seaborn as sns

# this dataset is already installed with seaborn
penguins = sns.load_dataset("penguins")

# function to remove missing values
penguins.dropna(inplace=True)

print(type(penguins))

penguins
```




```
<class 'pandas.core.frame.DataFrame'>
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
5	Adelie	Torgersen	39.3	20.6	190.0	3650.0	Male
...	...	...	...	...	...	...	...
338	Gentoo	Biscoe	47.2	13.7	214.0	4925.0	Female
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Female
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Female
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

333 rows × 7 columns

How to load the data

```
import seaborn as sns

# this dataset is already installed with seaborn
penguins = sns.load_dataset("penguins")

# function to remove missing values
penguins.dropna(inplace=True)

print(type(penguins))

penguins
```

?

```
<class 'pandas.core.frame.DataFrame'>
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
5	Adelie	Torgersen	39.3	20.6	190.0	3650.0	Male
...	...	...	...	...	...	...	...
338	Gentoo	Biscoe	47.2	13.7	214.0	4925.0	Female
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Female
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Female
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

333 rows × 7 columns

the data handling library in Python

Data is by default formatted as a **pandas dataframe** which works neatly together with Jupyter Notebook and Seaborn

Conversion between DataFrames and dictionaries

```
# make a sample dataset
data = {"column 1": [1, 4, 2, 4], "column 2": ["apple", "banana", "pineapple", "kiwi"]}

# convert to a pandas dataframe
dataframe = pd.DataFrame.from_dict(data)

dataframe
```

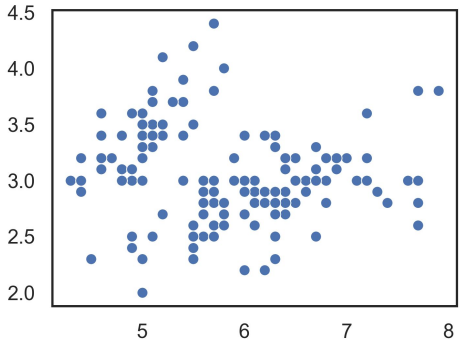
	column 1	column 2
0	1	apple
1	4	banana
2	2	pineapple
3	4	kiwi

```
# conversion of pandas dataframes back to dictionaries, the orient=list ensures
# that our dictionary values are formatted as lists again
data = dataframe.to_dict(orient="list")

print(data)
```

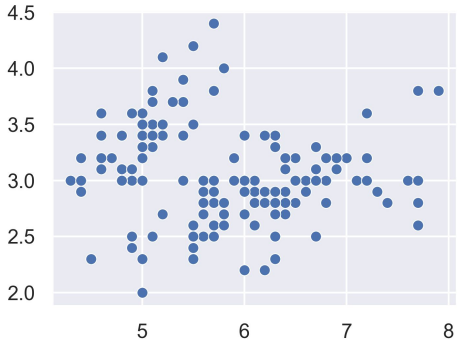
```
{'column 1': [1, 4, 2, 4], 'column 2': ['apple', 'banana', 'pineapple', 'kiwi']}
```

Get ready for plotting

```
import seaborn as sns
import matplotlib.pyplot as plt

# set the default figure size and resolution (dpi) and
# implicitly enable the seaborn plotting style
sns.set_theme(rc={"figure.dpi": 120, "figure.figsize": (8, 6)})
```

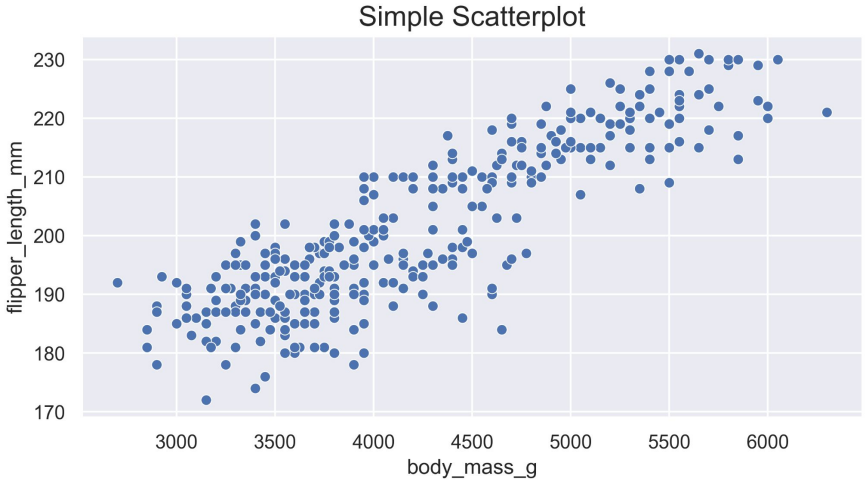


style



Our first scatterplot

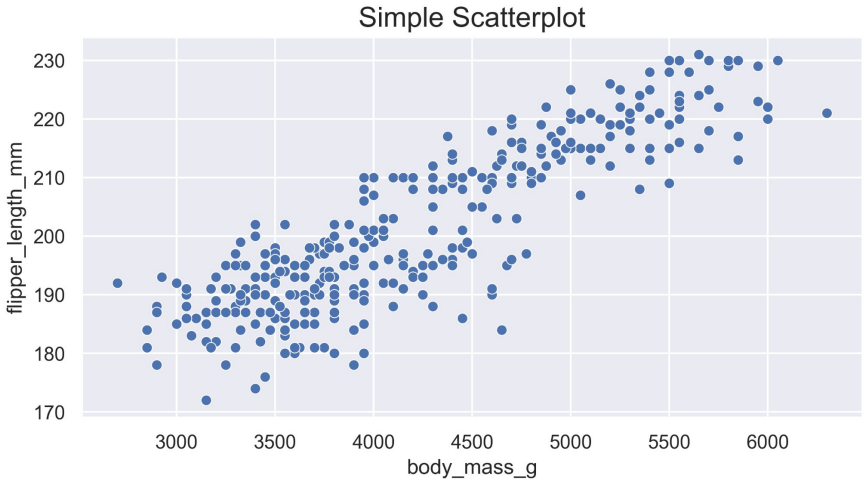
```
sns.scatterplot(data=penguins, x="body_mass_g", y="flipper_length_mm")

plt.title("Simple Scatterplot", fontsize=16)

plt.show()
```



State-based plotting

```
sns.scatterplot(data=penguins, x="body_mass_g", y="flipper_length_mm")

plt.title("Simple Scatterplot", fontsize=16)

plt.show()
plt.show()
```

this second call doesn't do anything



In **state-based plotting** we do not store our plot in any kind of variable or object and Python "forgets" it directly after drawing it.

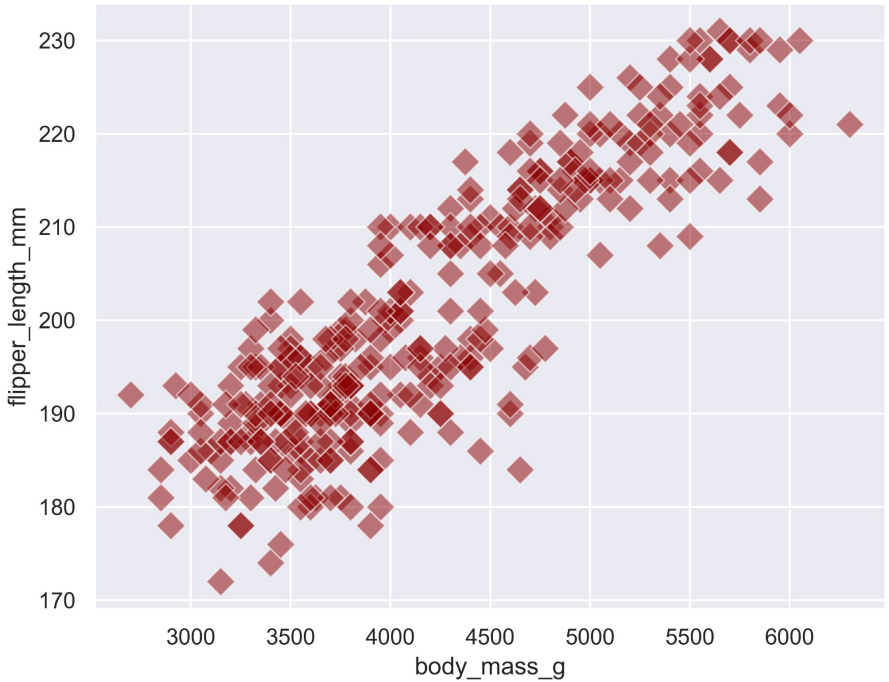
There is also object-oriented plotting but we won't use it in this course.

Options for customization

```
# alternative figure size
plt.figure(figsize = (7, 5.5))

sns.scatterplot(
    data=penguins,
    x="body_mass_g",
    y="flipper_length_mm",
    marker="D",
    color="darkred",
    alpha=0.5,
    s=10 ** 2,
)

plt.show()
```

see here for a list of available markers:

https://matplotlib.org/stable/api/markers_api.html#module-matplotlib.markers

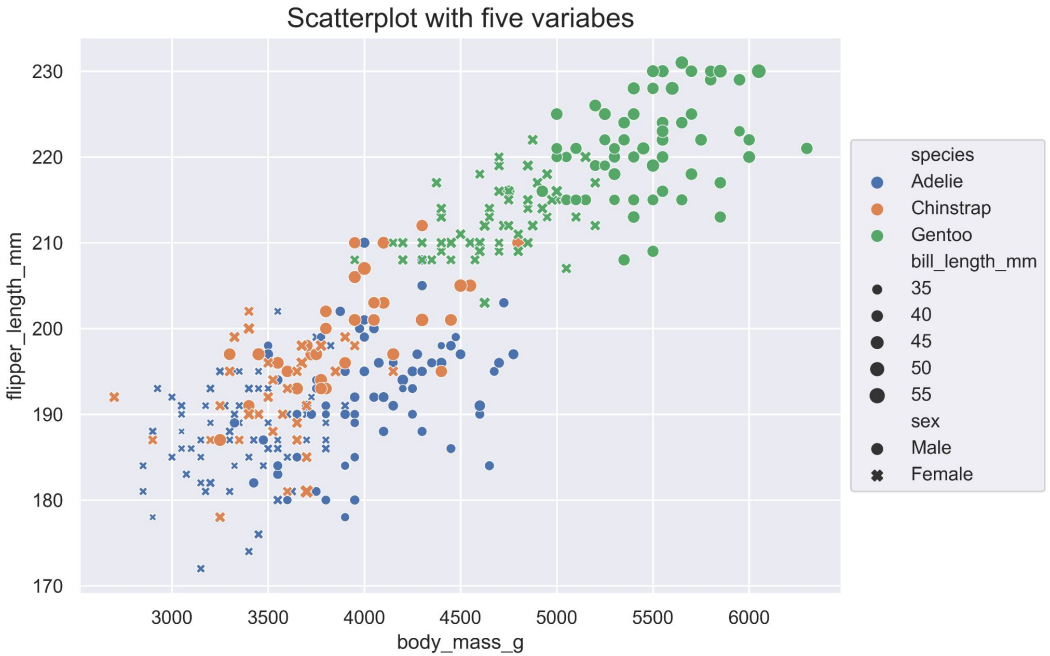


Four variables in one plot

```
sns.scatterplot(
    x="body_mass_g",
    y="flipper_length_mm",
    hue="species",
    style="sex",
    size="bill_length_mm",
    data=penguins
)

plt.title("Scatterplot with five variabes", fontsize=16)

plt.legend(loc="center left", bbox_to_anchor=(1, 0.5))

plt.show()
```

this moves the legend outside the plot, check out the link below for a detailed explanation

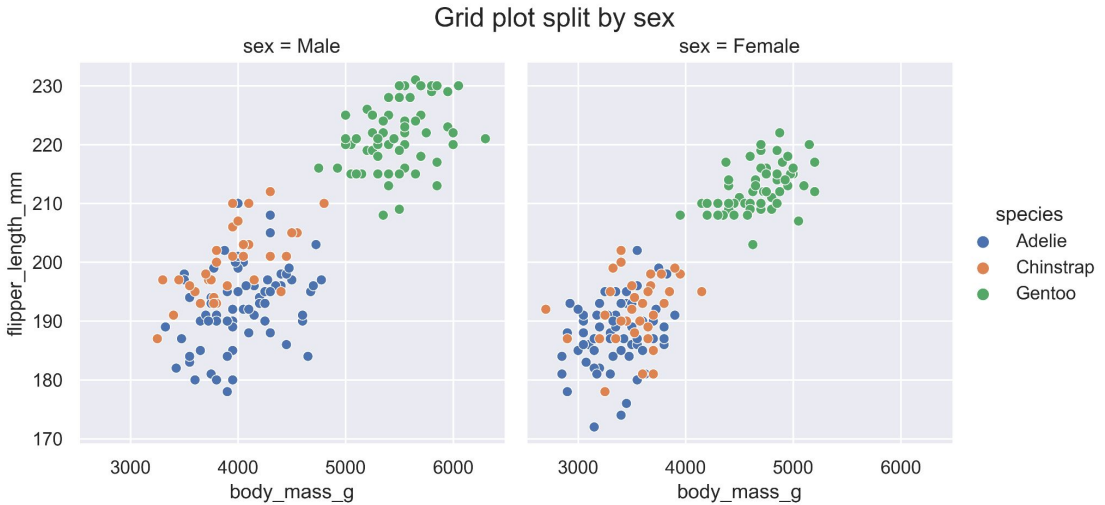
https://jdhao.github.io/2018/01/23/matplotlib-legend-outside-of-axes/



All plots have corresponding GridPlots

```
sns.relplot(
    data=penguins,
    x="body_mass_g",
    y="flipper_length_mm",
    hue="species",
    col="sex",
    aspect=5 / 5,
    height=4
)

# command to set the title for a grid plot
plt.suptitle("Grid plot split by sex", y=1.02, fontsize=16)

plt.show()
```

split by a variable

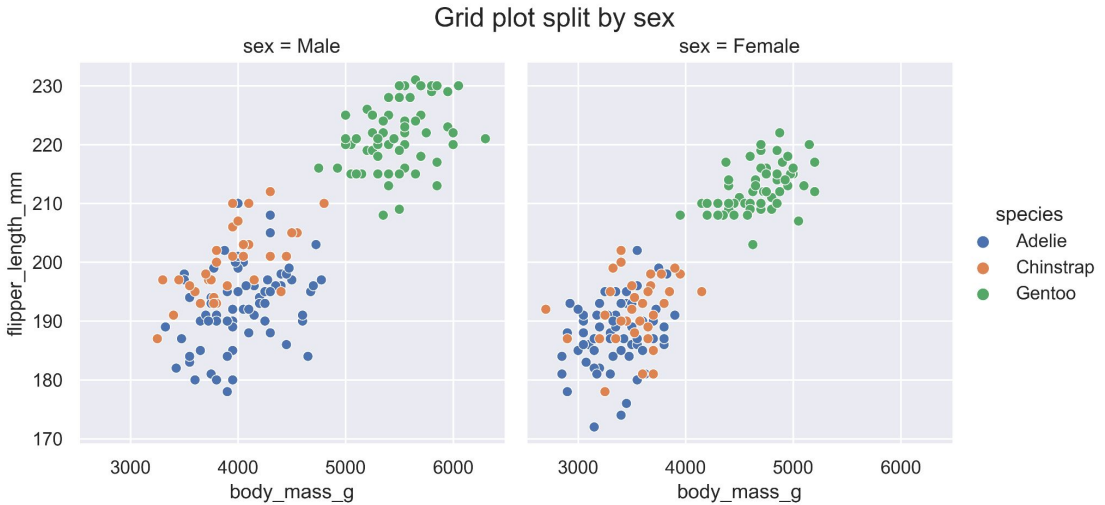


Gridplots: technicalities

```
sns.relplot(
    data=penguins,
    x="body_mass_g",
    y="flipper_length_mm",
    hue="species",
    col="sex",
    aspect=5 / 5,
    height=4
)

# command to set the title for a grid plot
plt.suptitle("Grid plot split by sex", y=1.02, fontsize=16)

plt.show()
```

you define subplot size directly in the function

you have to use a different title function and tweak it a bit

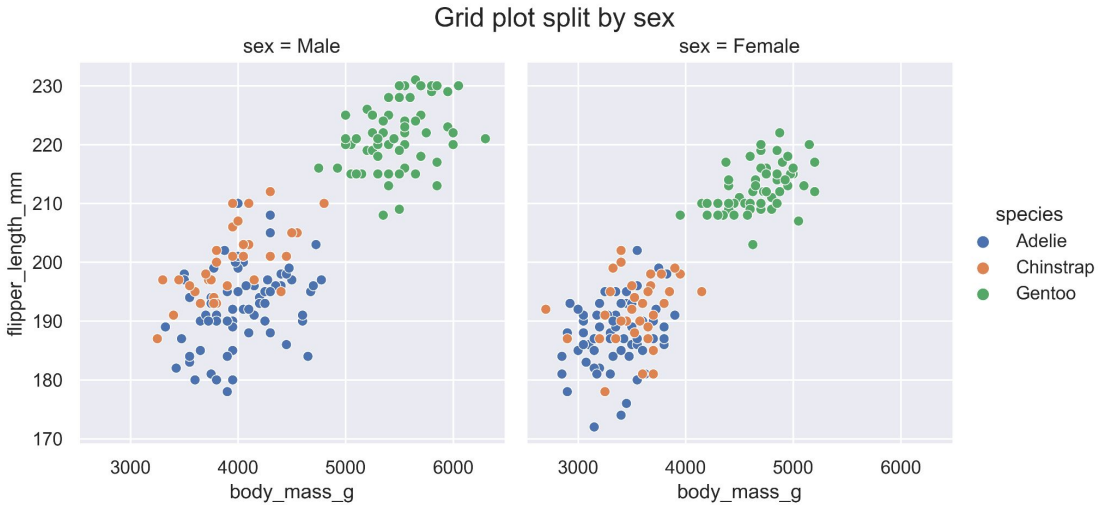


Gridplots: technicalities

```
sns.relplot(
    data=penguins,
    x="body_mass_g",
    y="flipper_length_mm",
    hue="species",
    col="sex",
    aspect=5 / 5,
    height=4
)

# command to set the title for a grid plot
plt.suptitle("Grid plot split by sex", y=1.02, fontsize=16)

plt.show()
```

you define subplot size directly in the function

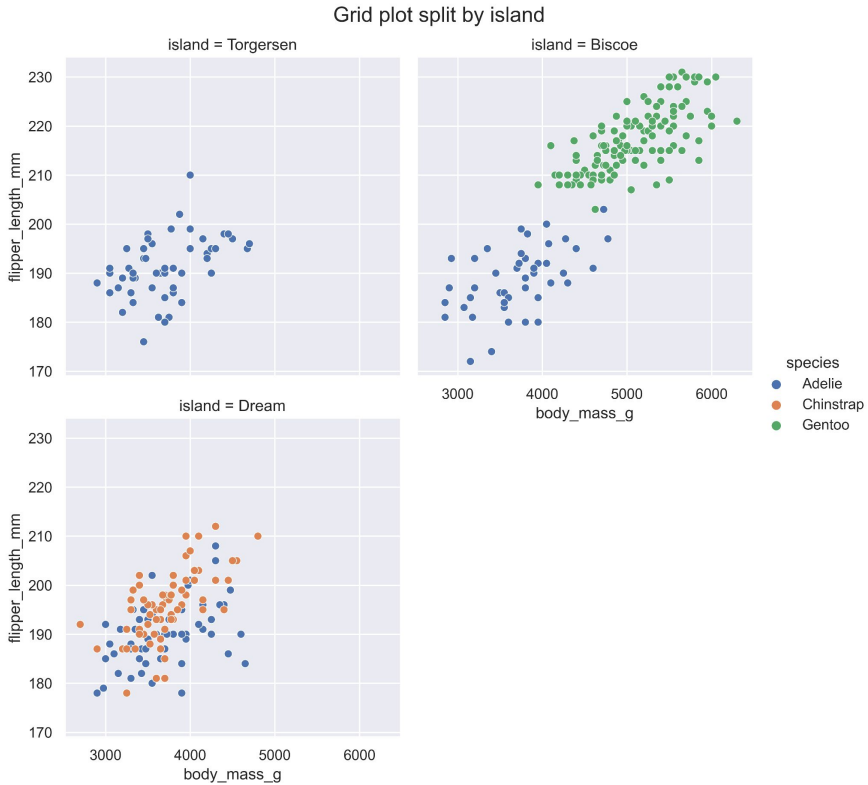
you have to use a different title function and tweak it a bit



Gridplots: technicalities

```
sns.relplot(
    data=penguins,
    x="body_mass_g",
    y="flipper_length_mm",
    hue="species",
    col="island",
    col_wrap=2,
    aspect=5 / 5,
    height=4
)

plt.suptitle("Grid plot split by island", y=1.02, fontsize=16)

plt.show()
```

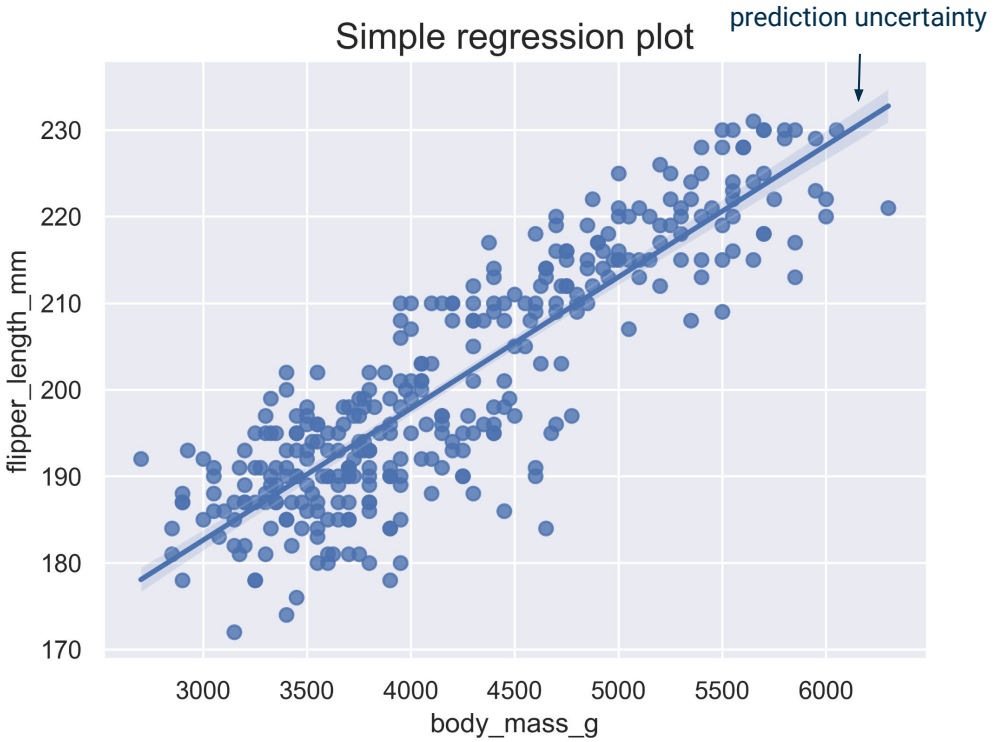
define the number of columns



Regression plots

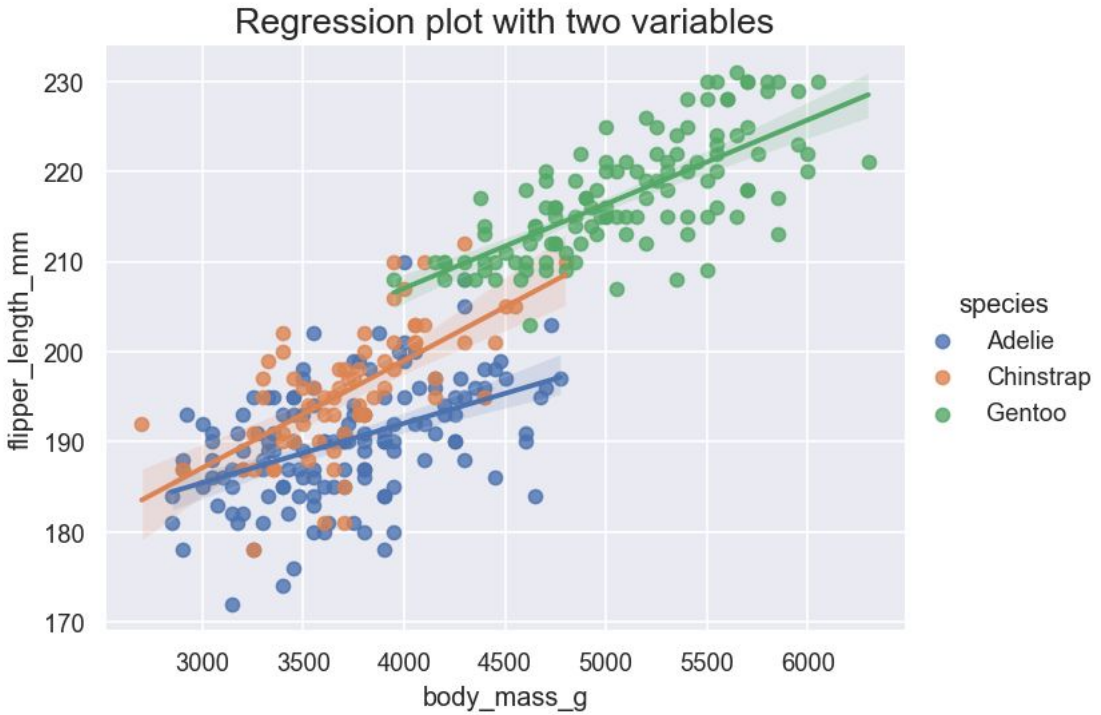
we use the gridplot function even for single plots because the single-plot function does not support the “hue” parameter (next slide)

```
sns.lmplot(
    data=penguins,
    x="body_mass_g",
    y="flipper_length_mm",
    height=4,
    aspect=6 / 4.5
)

plt.title("Simple regression plot", fontsize=16)
plt.show()
```



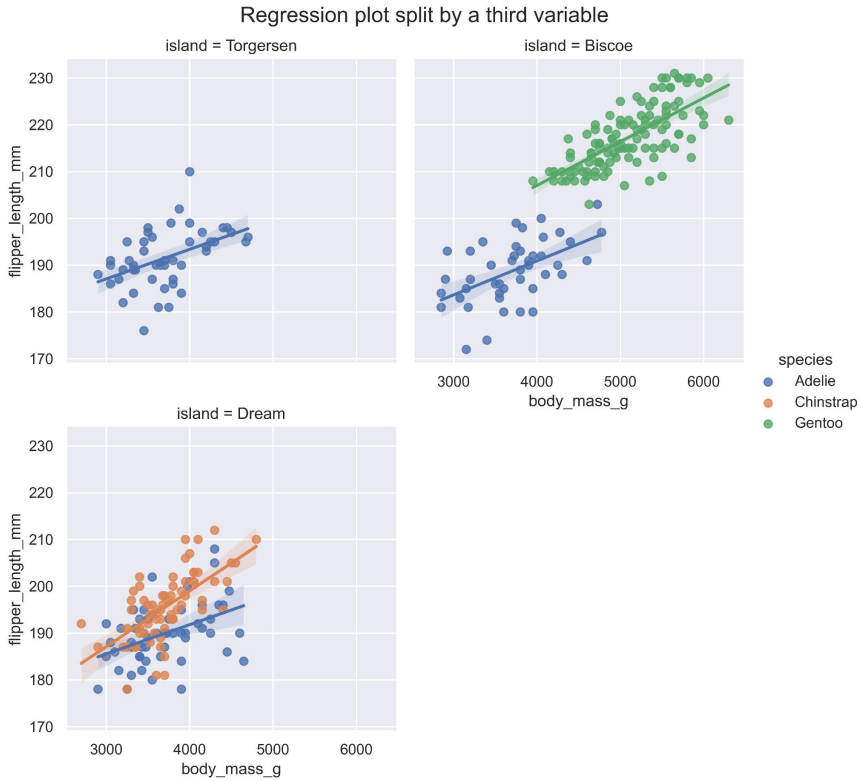
Regression plots

```
sns.lmplot(
    data=penguins,
    x="body_mass_g",
    y="flipper_length_mm",
    hue="species",
    height=4.5,
    aspect=6 / 4.5,
)

plt.title("Regression plot with two variables",
          fontsize=16)

plt.show()
```



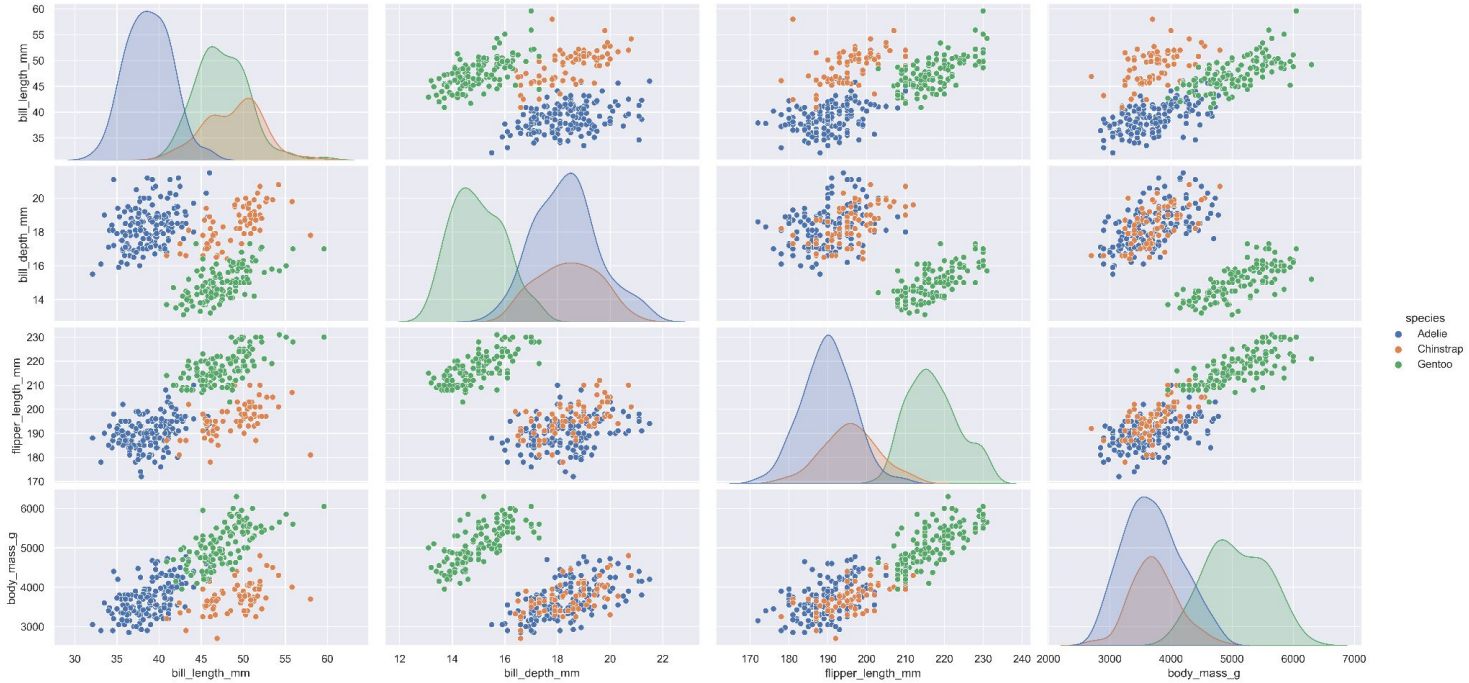
Regression plots

```
sns.lmplot(
    data=penguins,
    x="body_mass_g",
    y="flipper_length_mm",
    hue="species",
    col="island",
    height=4,
    aspect=5 / 5,
    col_wrap=2,
)

plt.suptitle("Regression plot split by a third variable",
             fontsize=16, y=1.02)
plt.show()
```

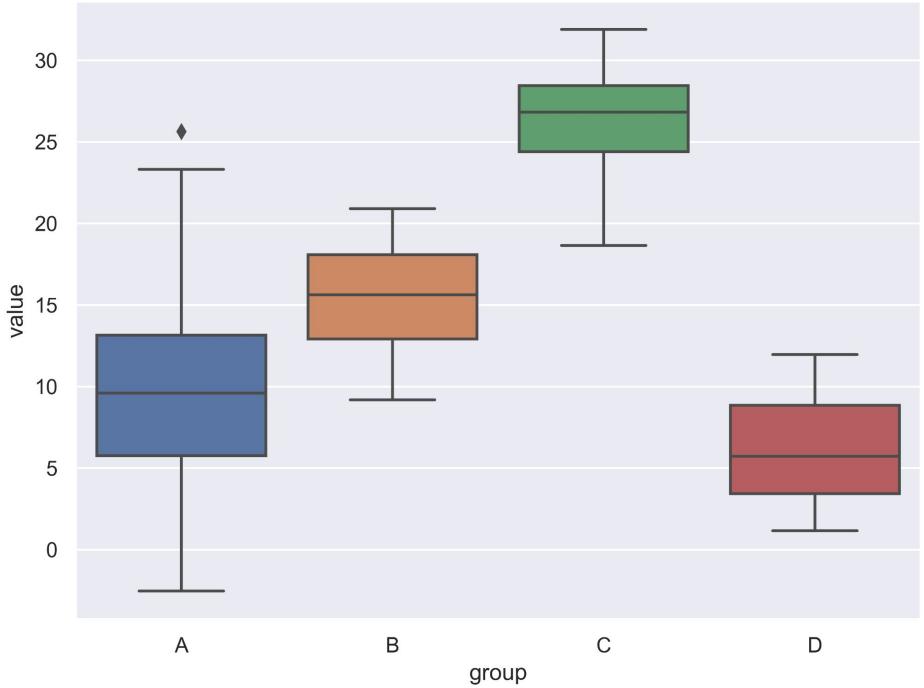


Pairplots: (most of) your data in one line of code

```
sns.pairplot(data=penguins, hue="species", aspect=2 / 1)
plt.show()
```

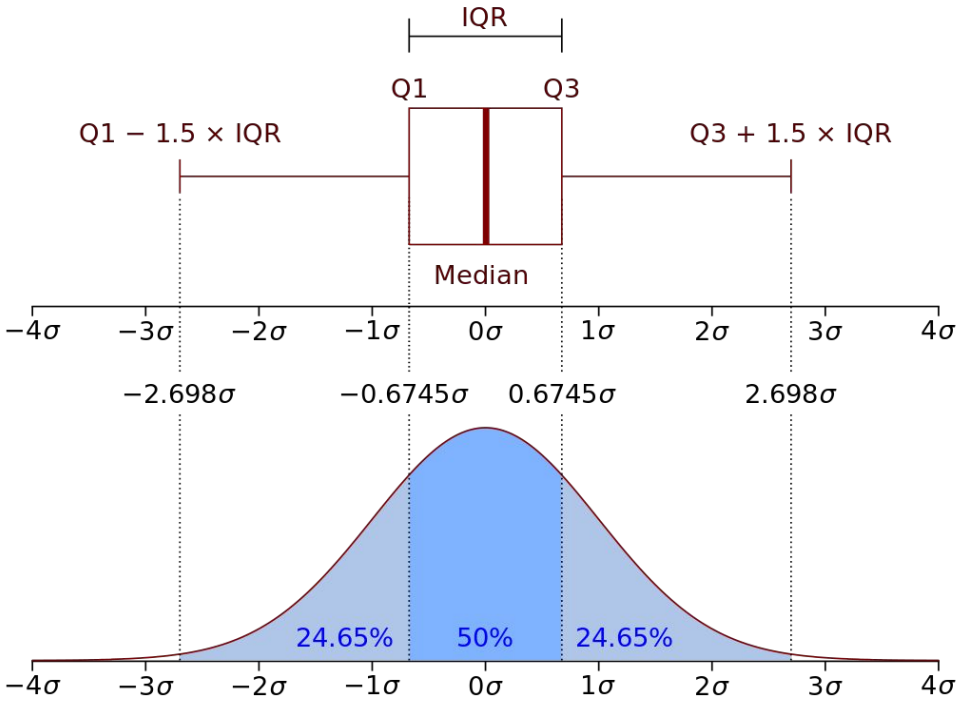


Boxplots



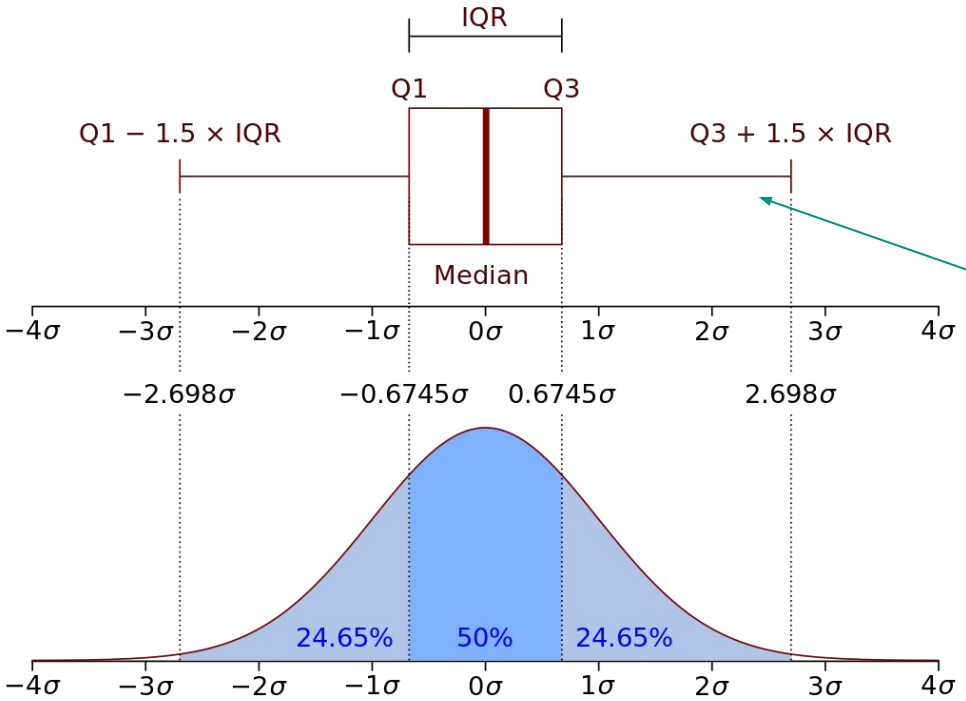
you might have seen these plots
but what do they really mean?

Boxplot structure



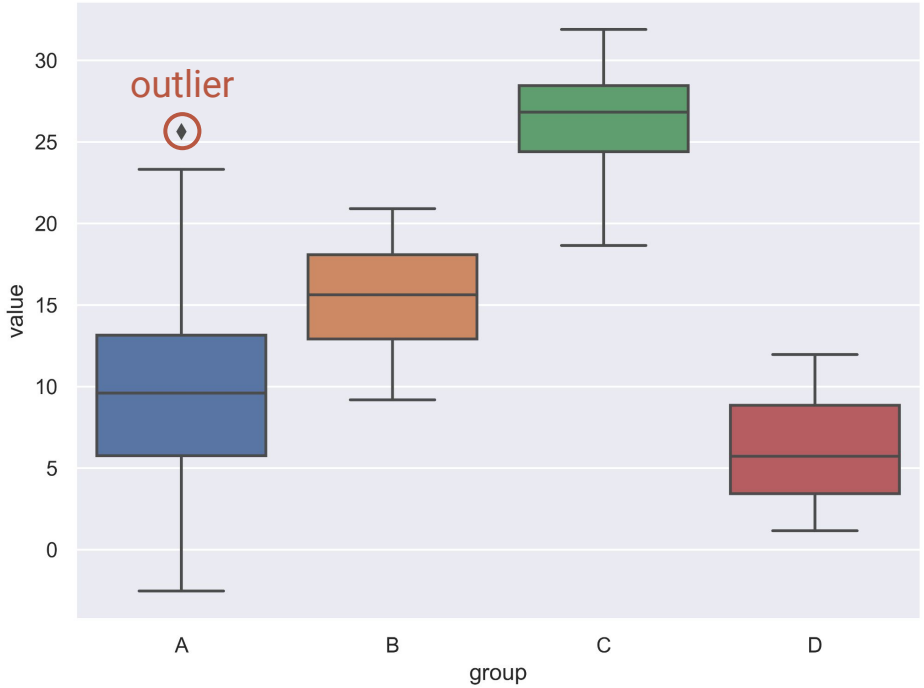
- **Median**
  the middle value of your data
- **1st Quartile (Q1)**
  25% of your data are below this value
- **3rd Quartile (Q3)**
  25% of your data are above this value
- **Interquartile range (IQR)**
  range between Q1 and Q3; contains the inner 50% of your data

Boxplot structure



- **Box**
  resembles the inner 50% of the data
- **Median-line**
  indicates the middle point of your data
- **Whiskers**
  lowest point within Q1 - 1.5*IQR
  highest point within Q3 - 1.5*IQR
  → spread of your data
- **Outliers**
  points outside the whiskers
  → extreme data points

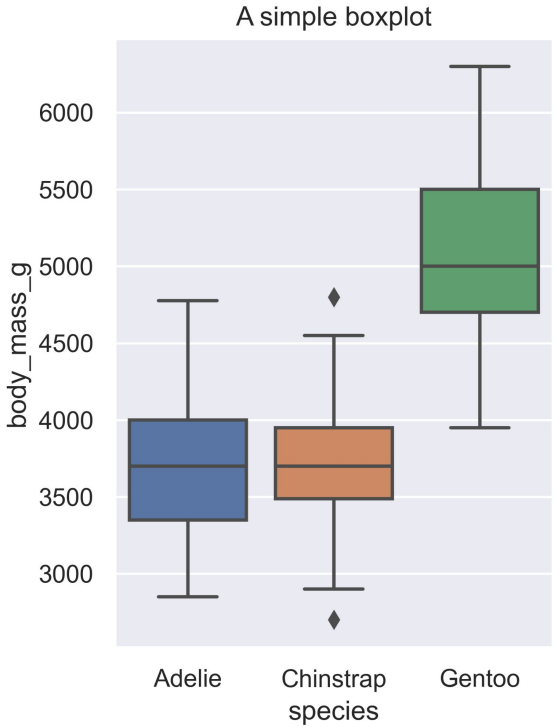
Boxplot structure



- **Box**
  resembles the inner 50% of the data
- **Median-line**
  indicates the middle point of your data
- **Whiskers**
  lowest point within Q1 - 1.5*IQR
  highest point within Q3 - 1.5*IQR
  → spread of your data
- **Outliers**
  points outside the whiskers
  → extreme data points

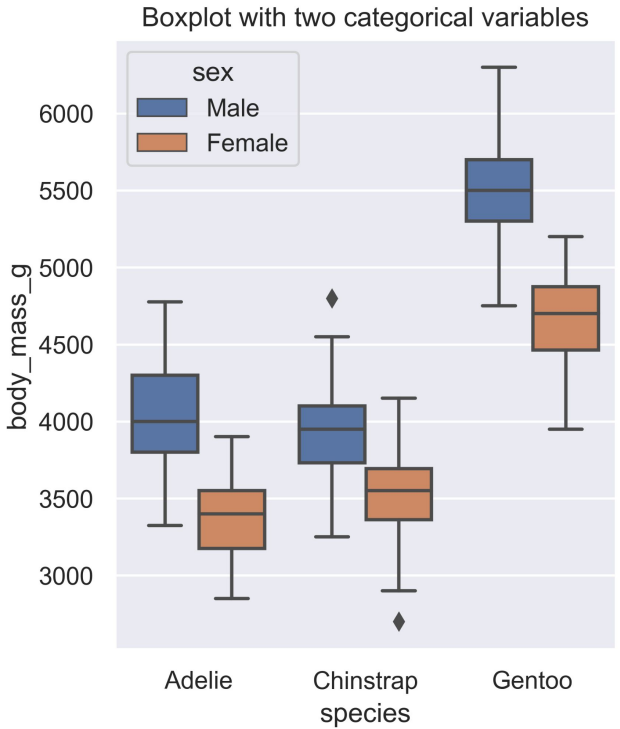
Boxplots in Seaborn

```
plt.figure(figsize=(3.5, 5))

sns.boxplot(
    data=penguins,
    x="species",
    y="body_mass_g"
)

plt.title("A simple boxplot")

plt.show()
```



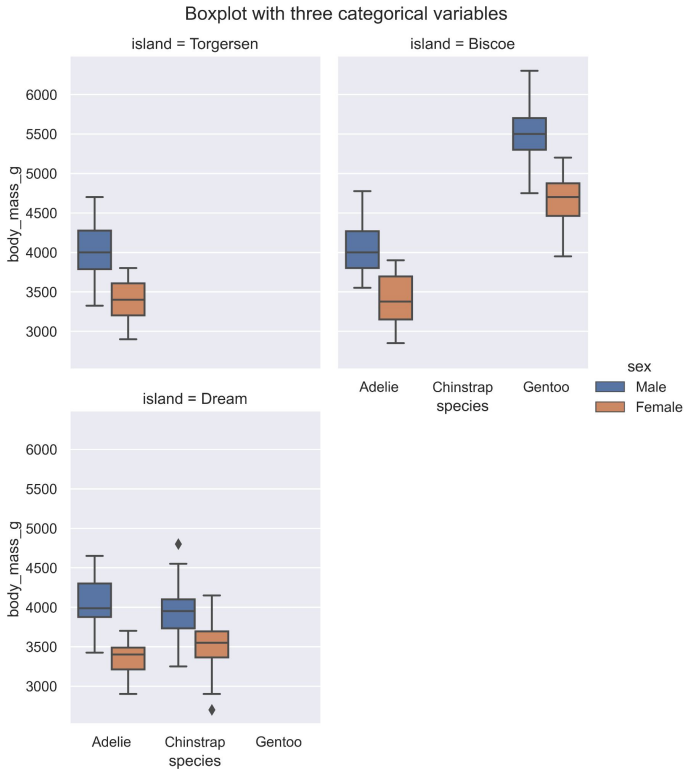
Boxplots with two categorical variables

```
plt.figure(figsize=(4, 5))

sns.boxplot(
    data=penguins,
    x="species",
    y="body_mass_g",
    hue="sex"
)

plt.title("Boxplot with two categorical variables")

plt.show()
```



Boxplot gridplots

```
sns.catplot(
    data=penguins,
    x="species",
    y="body_mass_g",
    col="island",
    hue="sex",
    col_wrap=2,
    height=4,
    aspect=4 / 5,
    kind="box"
)

plt.suptitle("Boxplot with three categorical variables", y=1.02)

plt.show()
```



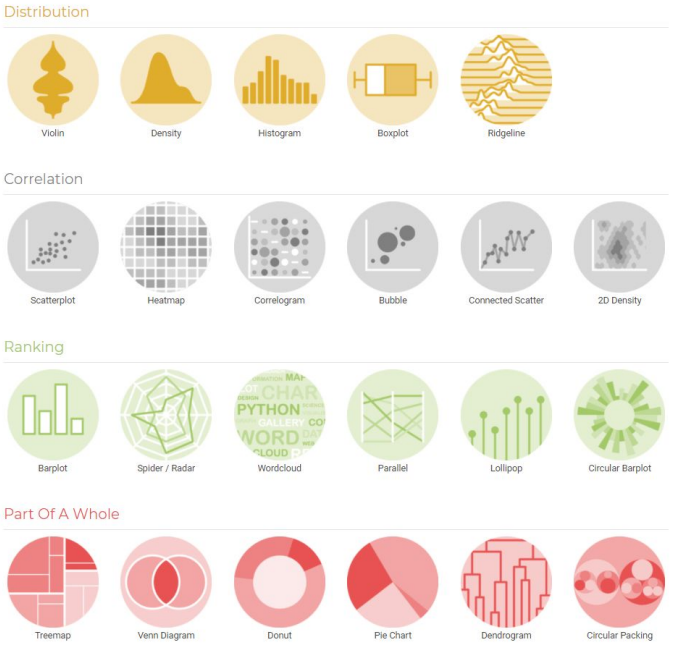
Finally saving your plot

path
resolution
makes sure nothing is cut off
allows backslashes

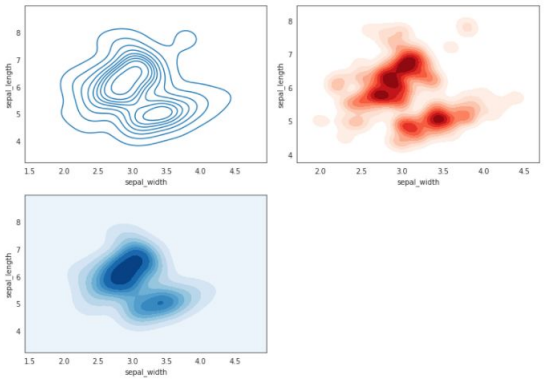
```
sns.scatterplot(data=penguins, x="body_mass_g", y="flipper_length_mm")
plt.title("Simple scatterplot", fontsize = 16)

# save your plot with an absolute path (will be different on your system)
plt.savefig(
    "C:/Users/jarosch/OneDrive/Studium/Master/Informatics in Biochemistry/Pictures/simple_scatterplot.png",
    dpi=300,
    bbox_inches="tight"
)

# Windows users: save your plot with an absolute path with backslashes
plt.savefig(
    r"C:\Users\jarosch\OneDrive\Studium\Master\Informatics in Biochemistry\Pictures\simple_scatterplot2.png",
    dpi=100,
    bbox_inches="tight"
)
```

Tip: Plot galleries – copy paste heaven!

https://www.python-graph-gallery.com/ or https://seaborn.pydata.org/examples/index.html




```
# libraries & dataset
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset('iris')

# set seaborn style
sns.set_style("white")

# Basic 2D density plot
sns.kdeplot(x=df.sepal_width, y=df.sepal_length)
plt.show()

# Custom the color, add shade and bandwidth
sns.kdeplot(x=df.sepal_width, y=df.sepal_length, cmap="Reds", shade=True, bw_adjust=.5)
plt.show()

# Add thresh parameter
sns.kdeplot(x=df.sepal_width, y=df.sepal_length, cmap="Blues", shade=True, thresh=0)
plt.show()
```



Introduction to plotting
Exercises

Dataset

For this exercise we move away from the penguins and look at the integrated restaurant tips dataset. This dataset contains data around the tips in a US restaurant collected over two and a half months of time:

- **total_bill**
- **tip**
- **sex**: sex of the person paying for the meal
- **smoker**: whether there was a smoker in the group
- **day**
- **time**
- **size**: size of the group

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

Exercise 1

Also remember that you should give each plot a title of your choice in the following exercises.

Draw a scatterplot plotting the total bill (x-axis) against the tip (y-axis) and give a different color to each day.

Exercise 2

Visually determine whether the trend-rate of tip increase over total bill increase is stronger for smokers or non-smokers using a linear regression plot.

Note: There seems to be a current specific bug with Jupyter Notebook and regression plots where the x-axis limits are set too tight and some points of your plot appear to be cut off. If you encounter this bug you can adjust the x-limits manually with `plt.gca().set(xlim=(lower_limit, upper_limit))`

```
```

Exercise 3 - Bonus

On which days and at which time (Lunch/Dinner) would you expect better tips from males compared to females? Determine this using a boxplot-gridplot representation (`catplot()` function).

Big Exercise 1

Exercise 1

Exercise 1

For this exercise, you will recreate the cellular RNA polymerase. Write a function called `RNA_polymerase()` that forms the complementary sequence to a DNA strand in RNA (output should be a string). Below you can already a find an example input and output sequence on which you can test your function. When your function is complete, you should also proof in code that your output sequence is equal to the supplied output sequence.

In [1]:
```
1 input_sequence = "ACATTTGCTTCTGACACAACTGTGTTCACTAGCAACCTCAAACAGACACCATGGTGCATCTGACTCCTGAGGAGAAGTCTGCCGTTACTGCCCTGTGC
```

Exercise 2

In the next step, you should translate the RNA sequence to protein in a function called Ribosome(). Remember that translation only starts at the start codon(`AUG`). And tranlation ends at the stop codon (`UAG` , `UGA` , `UAA`). You should use the codon table below.

```
codon_table = {
      'AUA':'I', 'AUC':'I', 'AUU':'I', 'AUG':'M',
      'ACA':'U', 'ACC':'U', 'ACG':'U', 'ACU':'U',
      'AAC':'N', 'AAU':'N', 'AAA':'K', 'AAG':'K',
      'AGC':'S', 'AGU':'S', 'AGA':'R', 'AGG':'R',
      'CUA':'L', 'CUC':'L', 'CUG':'L', 'CUU':'L',
      'CCA':'P', 'CCC':'P', 'CCG':'P', 'CCU':'P',
      'CAC':'H', 'CAU':'H', 'CAA':'Q', 'CAG':'Q',
      'CGA':'R', 'CGC':'R', 'CGG':'R', 'CGU':'R',
      'GUA':'V', 'GUC':'V', 'GUG':'V', 'GUU':'V',
      'GCA':'A', 'GCC':'A', 'GCG':'A', 'GCU':'A',
      'GAC':'D', 'GAU':'D', 'GAA':'E', 'GAG':'E',
      'GGA':'G', 'GGC':'G', 'GGG':'G', 'GGU':'G',
      'UCA':'S', 'UCC':'S', 'UCG':'S', 'UCU':'S',
      'UUC':'F', 'UUU':'F', 'UUA':'L', 'UUG':'L',
      'UAC':'Y', 'UAU':'Y', 'UAA':'_', 'UAG':'_',
      'UGC':'C', 'UGU':'C', 'UGA':'_', 'UGG':'W',
   }
```

Exercise 3

Exercise 3: Bonus

Write a function named `amino_acid_counter()` that accepts a protein sequence as input and returns a dictionary with the amino acids as keys and the count how many times they occur in the sequence as value. If you want, you can make use of the `Counter()` function from Python's base library `collections`.

Big Exercise 2

Exercise 1

You are given a file with physiological data from indian liver disease patients mixed together with healthy individuals (`additional_data/liver_disease.csv`). The "Dataset" column determines which patients actually have liver disease with "1" indicating sick individuals and "2" indicating healthy individuals.

1. Check out the file
2. Read in the following data from the csv file into *one* dictionary.
   Make the "Dataset" column more informative by renaming it to "Disease" with the two values "yes" or "no" instead of 1 and 2. Also make sure that your numerical columns contain actual numbers and not just strings!

   - Age
   - Gender
   - Total Proteins
   - Albumin
   - Disease (in the file it's called `Dataset`)
     The dictionary should be structured like this:
     `{'Age': [65, 62, ....], 'Gender': ['Female', 'Male',...], 'Total_Proteins [g/dl]': [6.8, 7.5, ...], 'Albumin [g/dl]': [3.3, 3.2, ...], 'Disease': ['No', 'No', ...]}`
     Index position `0` should be the first row with data etc...

liver_disease.csv

```
1   ;Age;Gender;Total_Bilirubin [mg/dl];Direct_Bilirubin [mg/dl];Alkaline_Phosphatase [U/l];Alanine_Aminotransferase [U/l]
2   0;65;Female;0.7;0.1;187;16;18;6.8;3.3;0.9;1
3   1;62;Male;10.9;5.5;699;64;100;7.5;3.2;0.74;1
4   2;62;Male;7.3;4.1;490;60;68;7.0;3.3;0.89;1
5   3;58;Male;1.0;0.4;182;14;20;6.8;3.4;1.0;1
6   4;72;Male;3.9;2.0;195;27;59;7.3;2.4;0.4;1
7   5;46;Male;1.8;0.7;208;19;14;7.6;4.4;1.3;1
8   6;26;Female;0.9;0.2;154;16;12;7.0;3.5;1.0;1
9   7;29;Female;0.9;0.3;202;14;11;6.7;3.6;1.1;1
10  8;17;Male;0.9;0.3;202;22;19;7.4;4.1;1.2;2
11  9;55;Male;0.7;0.2;290;53;58;6.8;3.4;1.0;1
12  10;57;Male;0.6;0.1;210;51;59;5.9;2.7;0.8;1
13  11;72;Male;2.7;1.3;260;31;56;7.4;3.0;0.6;1
```

Exercise 2

Exercise 2

Use a linear regression plot do determine how Albumin levels correlate with age. Is there a difference between sick and healthy individuals?

(The code below converts your dictionary into a pandas dataframe that you can use for plotting)

```
import pandas as pd

# replace `your_dict` with your dictionary created in Exercise 1
dataframe = pd.DataFrame.from_dict(your_dict)
dataframe
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# this sets the darkgrid theme that we've been using before
sns.set_theme()

```

Exercise 3

Exercise 3

Your **female** friend Anna is worried about her raised albumin levels, as she found an analysis of the liver disease dataset online that showed the following boxplot. Can you calm her down by plotting a more accurate representation of the data?

Bonus: Also save your new plot to a folder of your choice.

For the future:

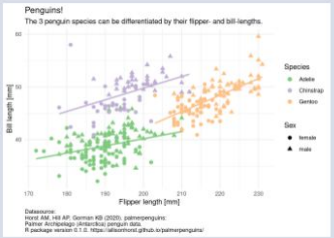
Course by Jannik Buhr in January

Teaching, Talks and Workshops

Learning by Sharing

Written by Jannik Buhr

2021



Introduction to data analysis with R WS21

Heidelberg University, Biochemistry Bachelor and Master (WS 21/22)

- Heidelberg University, INF 205

By Jannik Buhr in lecture series

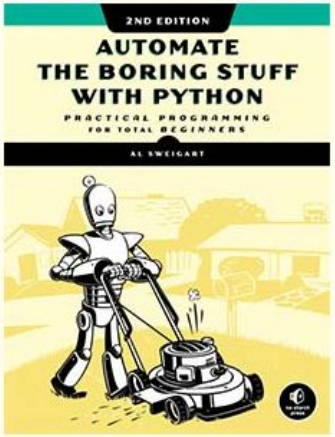
MATERIAL CODE

Automate the Boring Stuff with Python

free resource!

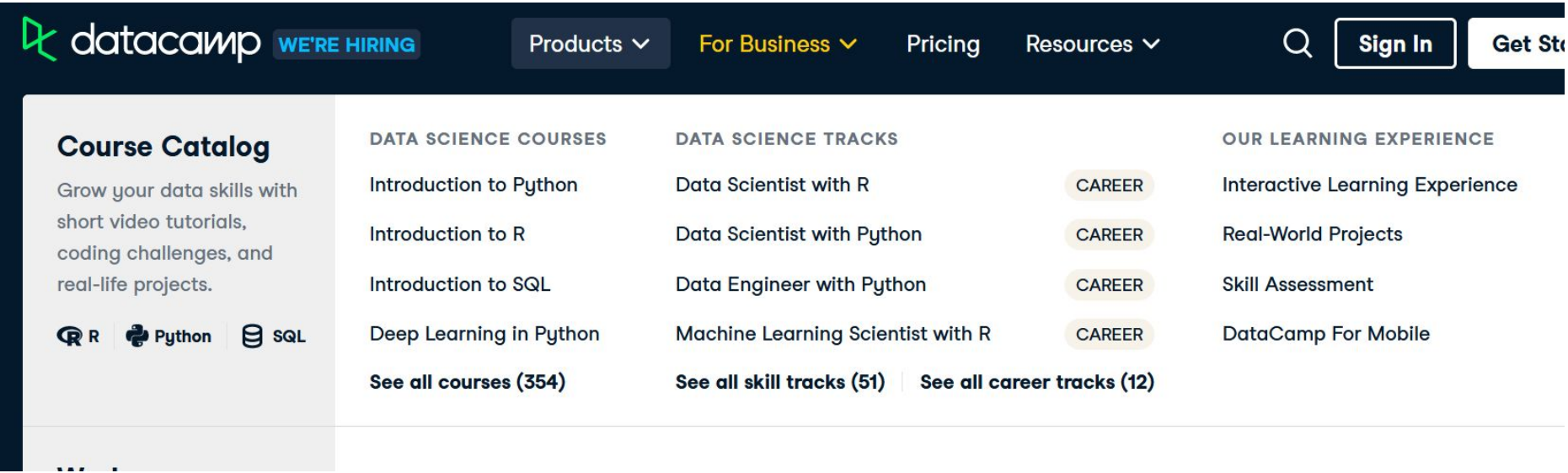
AUTOMATE THE BORING STUFF WITH PYTHON

By Al Sweigart. Over 285,000 copies sold. Free to read under a CC license.



https://automatetheboringstuff.com/

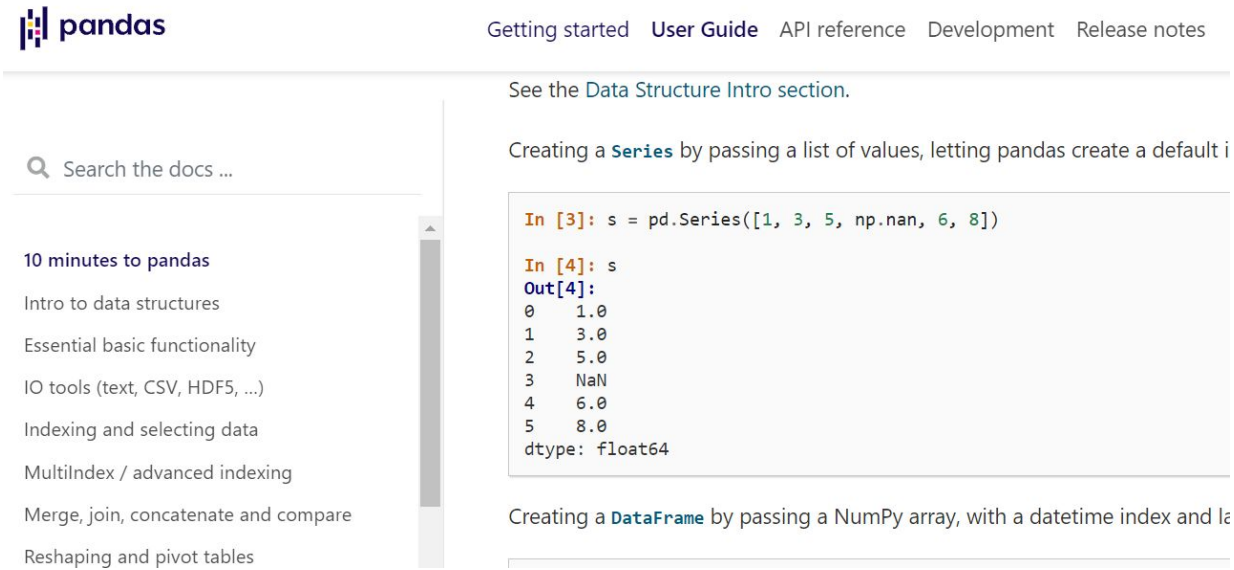
A 3 month long free course on DataCamp



free resource!
(for 3 months) https://docs.microsoft.com/en-us/visualstudio/subscriptions/vs-datacamp

Learn Pandas for easier data management

free resource!

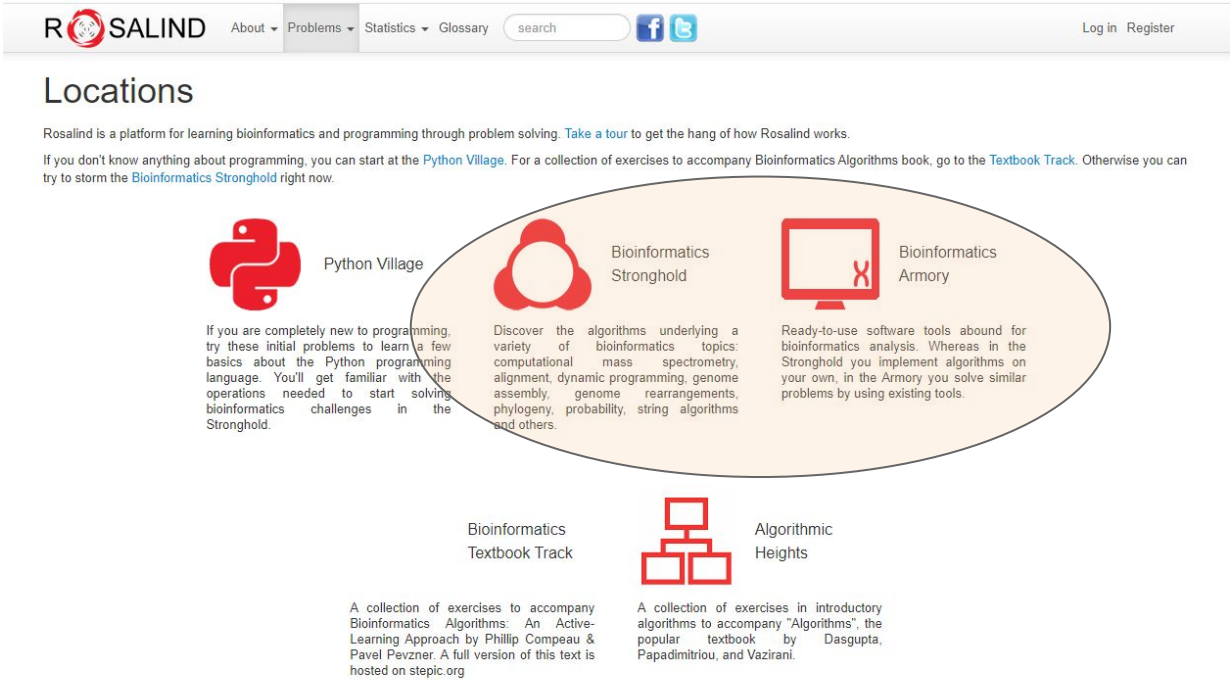


https://pandas.pydata.org/pandas-docs/stable/getting_started/tutorials.html

https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html

Take a bioinformatics challenge with ROSALIND

free resource!



http://rosalind.info/problems/locations/

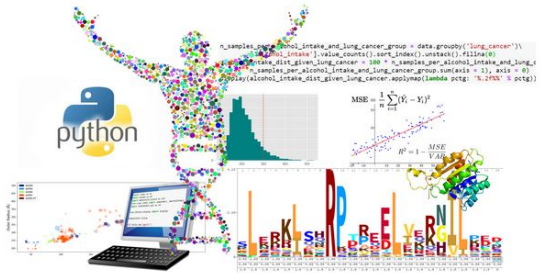
Elaborate introduction to bioinformatics

free resource!

misc moodle 2019/20

Home / My courses / Quantitative Biological Research with Python

Welcome to the online course: **Quantitative Biological Research with Python**



What is this course?

Quantitative Biological Research with Python is an **online course**, originally given at the **Hebrew University of Jerusalem**. The course is aimed for biology researchers (mostly graduate-level students) interested in conducting quantitative research and analyzing high-throughput data.

Teachers:

https://muddle2.cs.huji.ac.il/ru19/course/view.php?id=68