CEPLAS
Cluster of Excellence on Plant Sciences

Introduction to Python

QBio104

Material modified from M. Röttger and A. Schrader

Recap

Topics

- Variables, objects, data types
- Indexing
- Functions and methods
- Creating a function

Variables, objects, data types

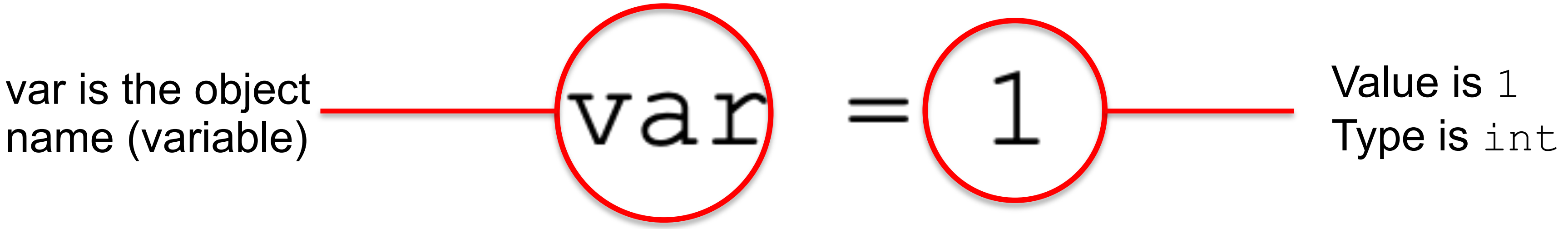
Definitions

”Object” is defined as the area of memory that holds the **“Values”** and is characterized by a **“Type”**

“Variable” is the name of an **”Object”**

“Type” specifies how to interpret the **“Values”** of an **“Object”**

“Values” are the actual data in an **”Object”**

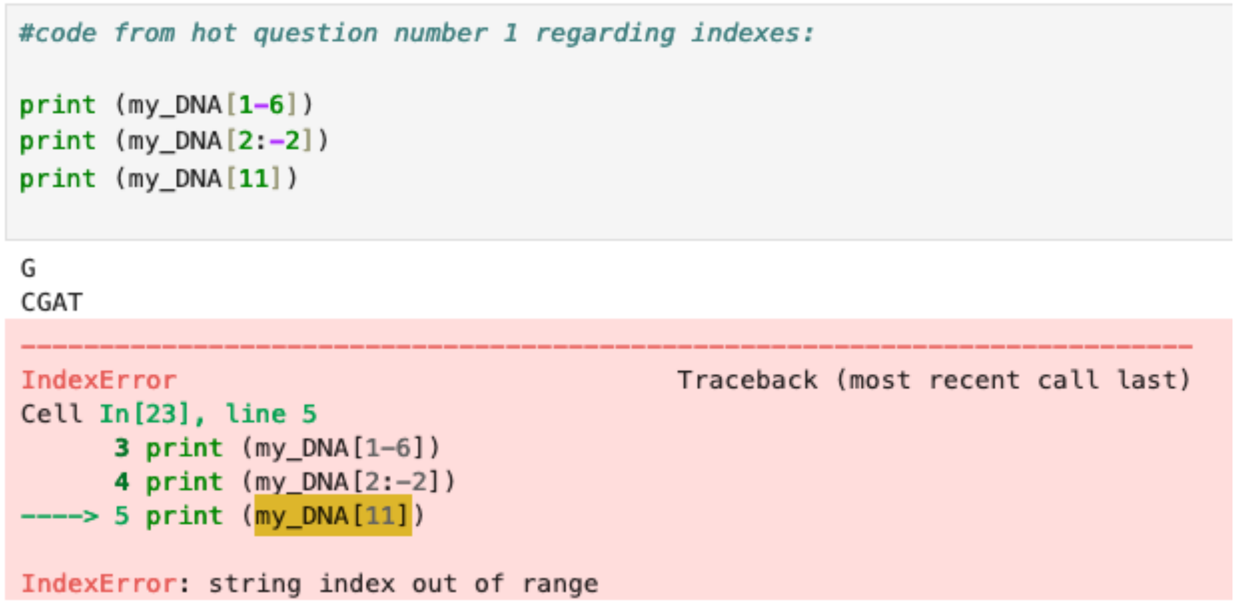


Indexing

Sequence object types can be accessed via indexing

- You can refer to one or more items in a sequence object type (strings, lists, tuples) by specifying the index using [] after the variable name. Note that the index, position, in a list or string, starts from 0
- To specify a range use `":"` excluding the second value
- Negative values return elements from the end starting from -1

+	0	1	2	3	4	5	6	7
str	A	T	C	G	A	T	C	G
-	-8	-7	-6	-5	-4	-3	-2	-1



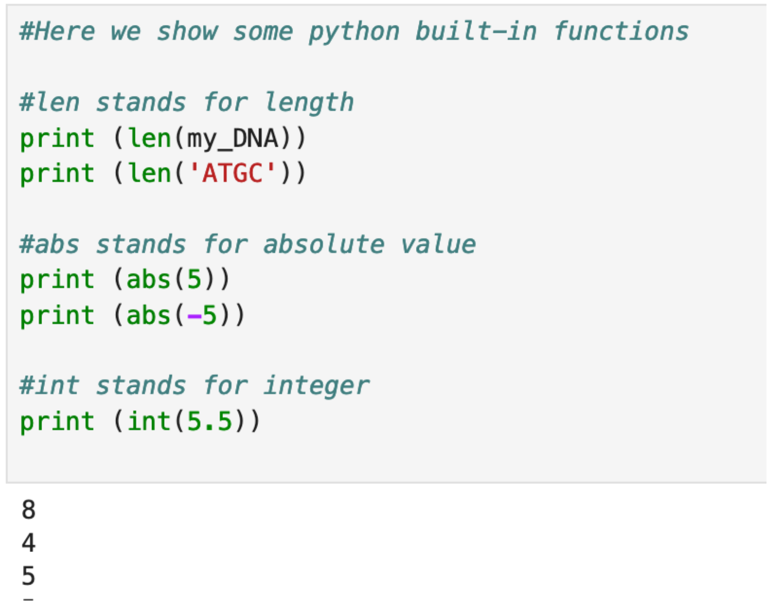
```
#code from hot question number 1 regarding indexes:

print (my_DNA[1-6])
print (my_DNA[2:-2])
print (my_DNA[11])
```

```
G
CGAT
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[23], line 5
      3 print (my_DNA[1-6])
      4 print (my_DNA[2:-2])
----> 5 print (my_DNA[11])

IndexError: string index out of range
```

Functions and methods

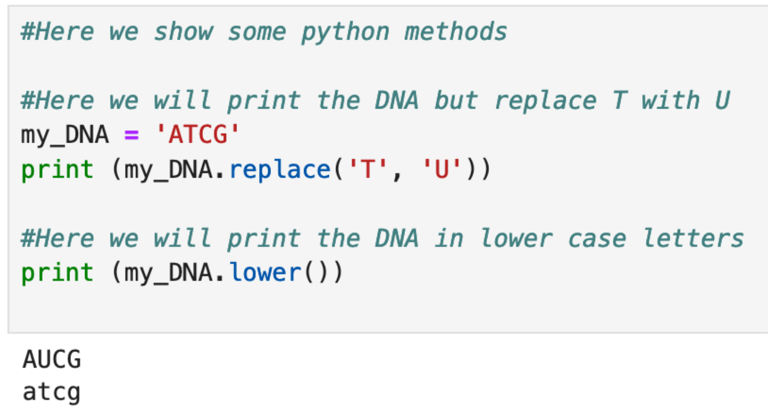
Pre-defined set of instructions to perform tasks

- Some are available all the time, others can be imported and you can define your own functions
- To use a **function** you first state the function name followed by the arguments within parenteses
- To use a **method**, you first write the object followed by a “**.**” followed by the method name, followed by the arguments within parenteses



```
#Here we show some python built-in functions

#len stands for length
print (len(my_DNA))
print (len('ATGC'))

#abs stands for absolute value
print (abs(5))
print (abs(-5))

#int stands for integer
print (int(5.5))
```

```
8
4
5
```



```
#Here we show some python methods

#Here we will print the DNA but replace T with U
my_DNA = 'ATCG'
print (my_DNA.replace('T', 'U'))

#Here we will print the DNA in lower case letters
print (my_DNA.lower())
```

```
AUCG
atcg
```

Creating a function

Automate repetitive tasks

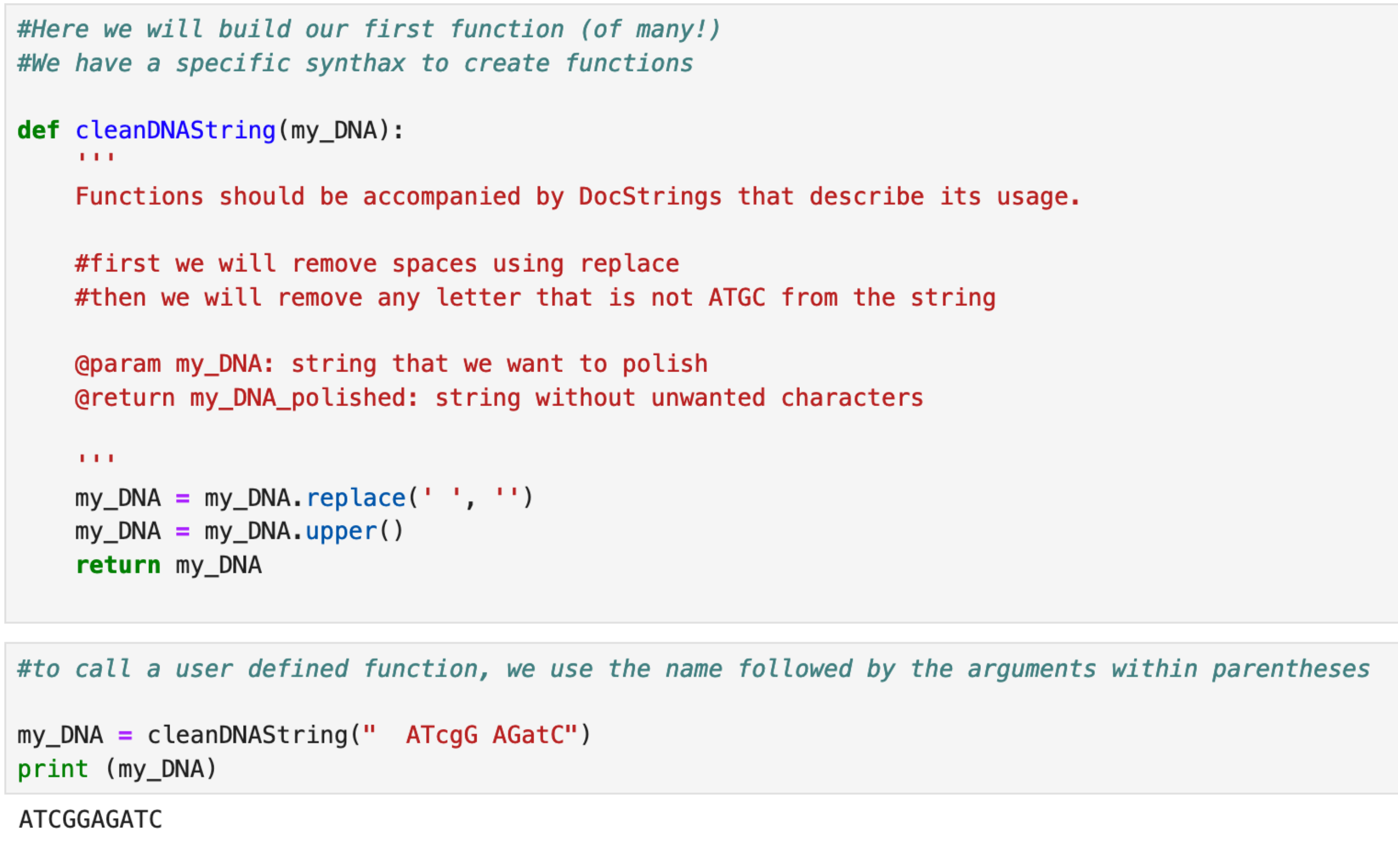
Beyond built-in functions, you can also build your own set of instructions that are called by a function.

We initiate a function using `def`

Followed by the name of the new function and the parameters that we need to execute the function and a colon `:`

The lines of code within a function will be “indented” (using `tab`)

At the end of a function, we use return to define what will come out of the function. Hint, it can be more than just one variable



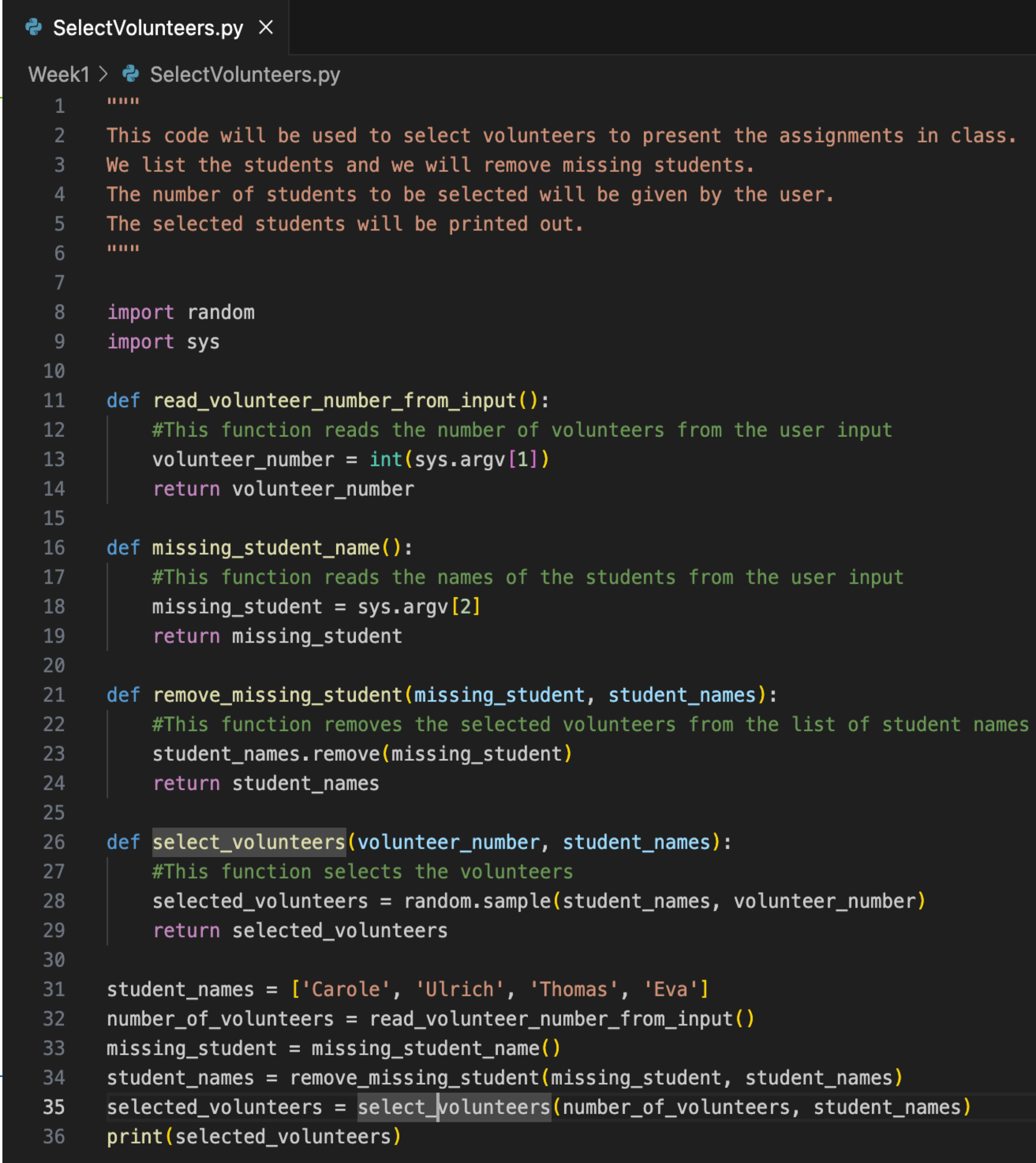
```
#Here we will build our first function (of many!)
#We have a specific synthax to create functions

def cleanDNAString(my_DNA):
    '''
    Functions should be accompanied by DocStrings that describe its usage.

    #first we will remove spaces using replace
    #then we will remove any letter that is not ATGC from the string

    @param my_DNA: string that we want to polish
    @return my_DNA_polished: string without unwanted characters

    '''
    my_DNA = my_DNA.replace(' ', '')
    my_DNA = my_DNA.upper()
    return my_DNA
```

```
#to call a user defined function, we use the name followed by the arguments within parentheses

my_DNA = cleanDNAString("  ATcgG AGatC")
print (my_DNA)
```

```
ATCGGAGATC
```

Creating and using functions

- The import section goes at the beginning
- We break down the code into functions
- Functions are created when running the program but do not “run” until “called”
- `sys.argv[0]` is the script name (`SelectVolunteers.py`)



```python
"""
This code will be used to select volunteers to present the assignments in class.
We list the students and we will remove missing students.
The number of students to be selected will be given by the user.
The selected students will be printed out.
"""

import random
import sys

def read_volunteer_number_from_input():
    #This function reads the number of volunteers from the user input
    volunteer_number = int(sys.argv[1])
    return volunteer_number

def missing_student_name():
    #This function reads the names of the students from the user input
    missing_student = sys.argv[2]
    return missing_student

def remove_missing_student(missing_student, student_names):
    #This function removes the selected volunteers from the list of student names
    student_names.remove(missing_student)
    return student_names

def select_volunteers(volunteer_number, student_names):
    #This function selects the volunteers
    selected_volunteers = random.sample(student_names, volunteer_number)
    return selected_volunteers

student_names = ['Carole', 'Ulrich', 'Thomas', 'Eva']
number_of_volunteers = read_volunteer_number_from_input()
missing_student = missing_student_name()
student_names = remove_missing_student(missing_student, student_names)
selected_volunteers = select_volunteers(number_of_volunteers, student_names)
print(selected_volunteers)
```

Introduction to python

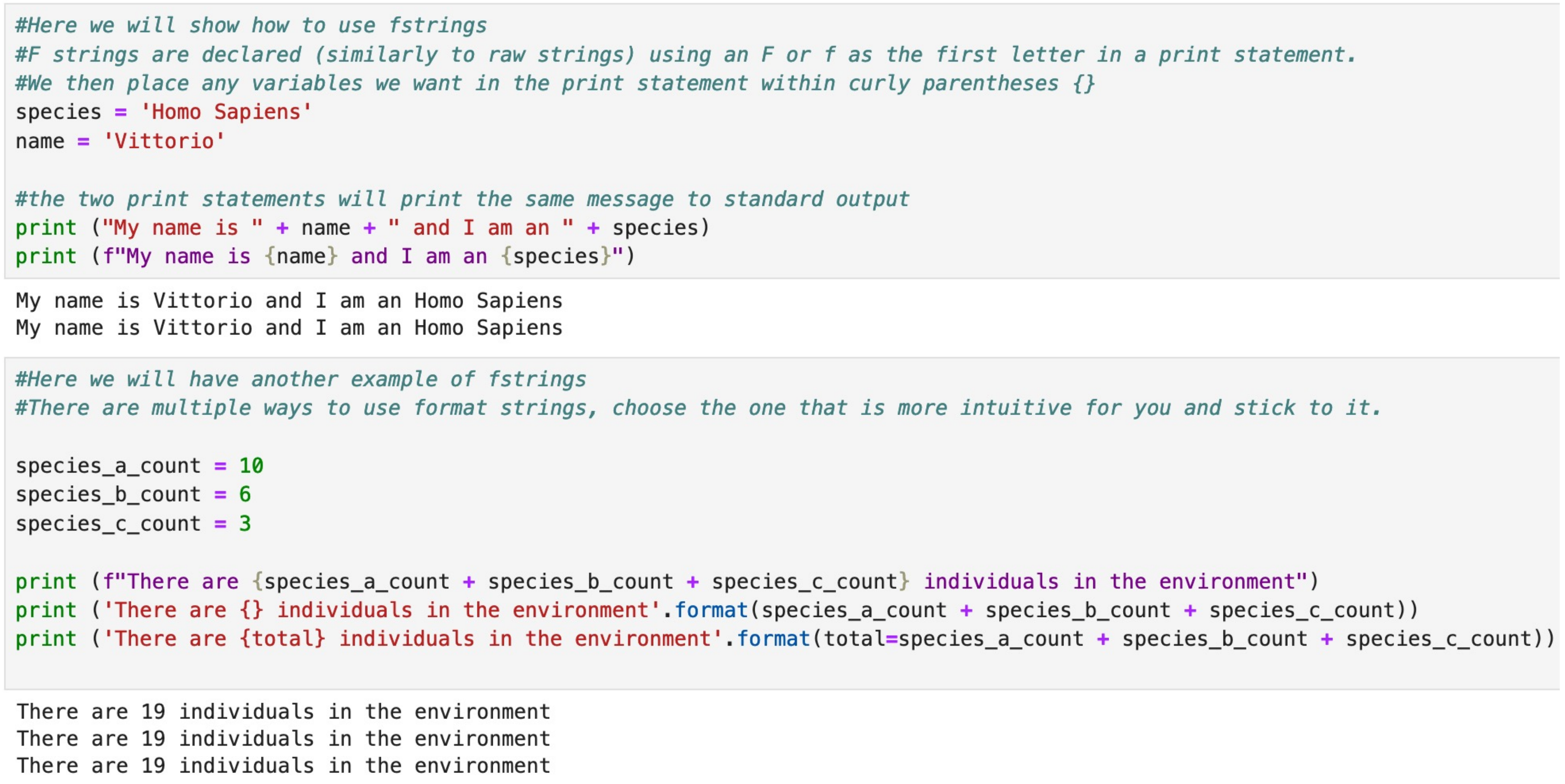
Topics

- Better print statements
- Search, find, strip, reverse
- Control structures
    - If
    - Else
    - elif
    - Nested
- Creating functions
    - Improve the student selection program
    - More programs

Better print statements

Formatted strings

- We already learned about metacharacters and raw strings
- There are multiple ways to format and include variables in print statements



```python
#Here we will show how to use fstrings
#F strings are declared (similarly to raw strings) using an F or f as the first letter in a print statement.
#We then place any variables we want in the print statement within curly parentheses {}
species = 'Homo Sapiens'
name = 'Vittorio'

#the two print statements will print the same message to standard output
print ("My name is " + name + " and I am an " + species)
print (f"My name is {name} and I am an {species}")
```

```
My name is Vittorio and I am an Homo Sapiens
My name is Vittorio and I am an Homo Sapiens
```

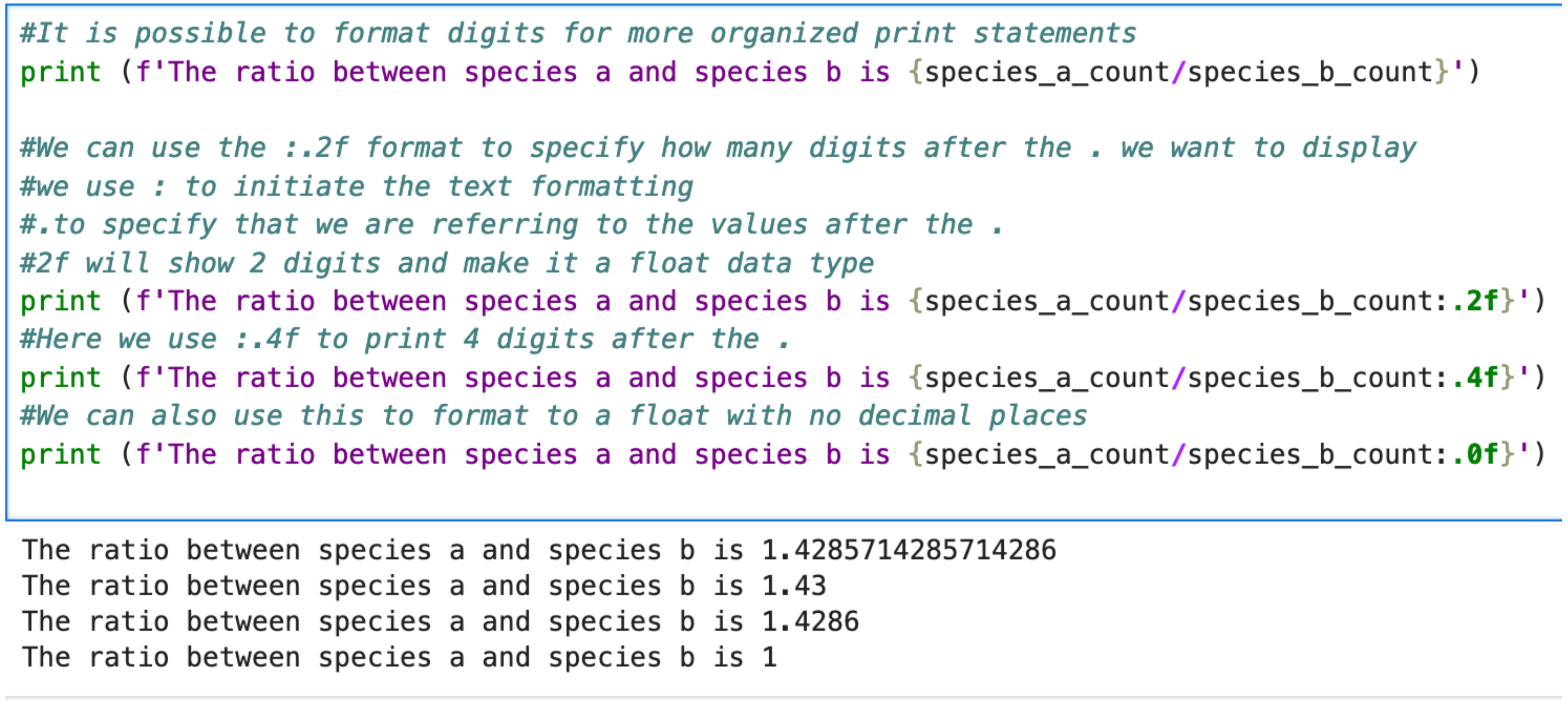
```python
#Here we will have another example of fstrings
#There are multiple ways to use format strings, choose the one that is more intuitive for you and stick to it.

species_a_count = 10
species_b_count = 6
species_c_count = 3

print (f"There are {species_a_count + species_b_count + species_c_count} individuals in the environment")
print ('There are {} individuals in the environment'.format(species_a_count + species_b_count + species_c_count))
print ('There are {total} individuals in the environment'.format(total=species_a_count + species_b_count + species_c_count))
```

```
There are 19 individuals in the environment
There are 19 individuals in the environment
There are 19 individuals in the environment
```

Formatted strings

An elegant way to display numbers in print statements

- Arithmetic operations may result in numbers that we want to format when printing
- We use the notation `:.2f where :` initiates the formatting `.2f` indicates the number of digits to display after the . in the float



```python
#It is possible to format digits for more organized print statements
print (f'The ratio between species a and species b is {species_a_count/species_b_count}')

#We can use the :.2f format to specify how many digits after the . we want to display
#we use : to initiate the text formatting
#.to specify that we are referring to the values after the .
#2f will show 2 digits and make it a float data type
print (f'The ratio between species a and species b is {species_a_count/species_b_count:.2f}')
#Here we use :.4f to print 4 digits after the .
print (f'The ratio between species a and species b is {species_a_count/species_b_count:.4f}')
#We can also use this to format to a float with no decimal places
print (f'The ratio between species a and species b is {species_a_count/species_b_count:.0f}')
```

```
The ratio between species a and species b is 1.4285714285714286
The ratio between species a and species b is 1.43
The ratio between species a and species b is 1.4286
The ratio between species a and species b is 1
```

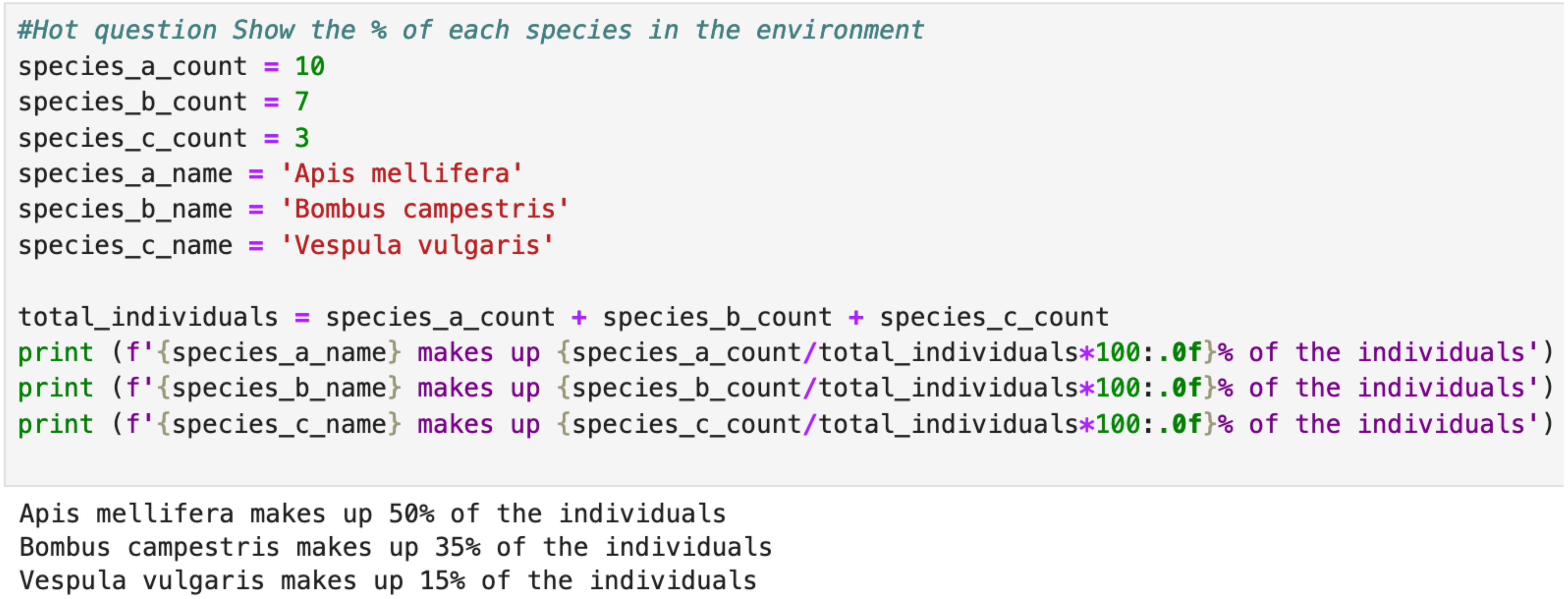
Hot question

Let’s make a print statement together

- Use as many formatting strategies as you can
- Starting from:
  - species_a_count = 10
    - species_a_name = ‘*Apis mellifera*’
  - species_b_count = 7
    - species_b_name = ‘*Bombus campestris*’
  - species_c_count = 3
    - species_c_name = *‘Vespula vulgaris’*
  - For simplicity, calculate the `total_individuals` variable beforehand
- Show the % of each species in the environment
  - Remember the float formatting `:.2f` etc..

Hot question

Answer (one of the possible ones)



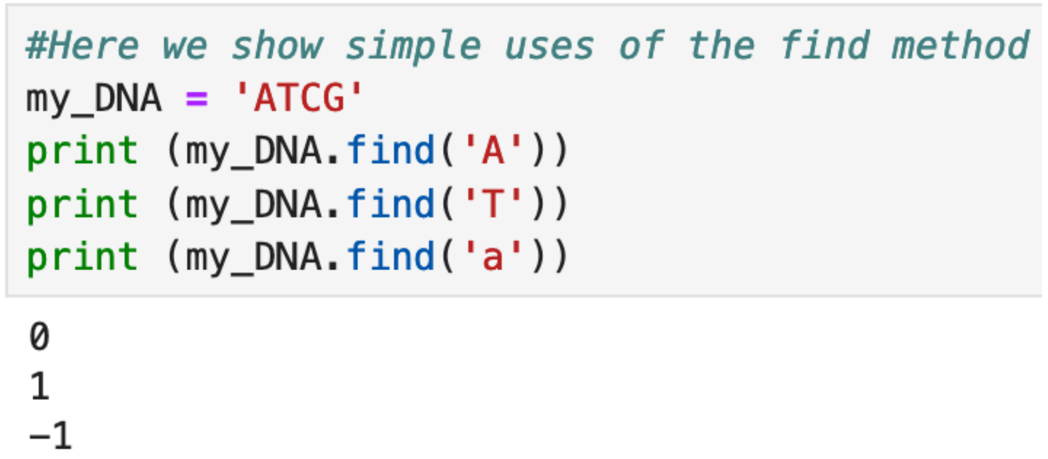
```python
#Hot question Show the % of each species in the environment
species_a_count = 10
species_b_count = 7
species_c_count = 3
species_a_name = 'Apis mellifera'
species_b_name = 'Bombus campestris'
species_c_name = 'Vespula vulgaris'

total_individuals = species_a_count + species_b_count + species_c_count
print (f'{species_a_name} makes up {species_a_count/total_individuals*100:.0f}% of the individuals')
print (f'{species_b_name} makes up {species_b_count/total_individuals*100:.0f}% of the individuals')
print (f'{species_c_name} makes up {species_c_count/total_individuals*100:.0f}% of the individuals')
```

```
Apis mellifera makes up 50% of the individuals
Bombus campestris makes up 35% of the individuals
Vespula vulgaris makes up 15% of the individuals
```

Text search

Searching for a substring using `find(…)`

- The method `str.find(`sub, start, end`)` returns the index of the first appearance of the substring sub in string str.
  - Index 0 means that sub was found starting at the first position in str.
  - Index 1 means it was found starting at position 2, and so on.
- If sub was not found in str, the method returns -1.



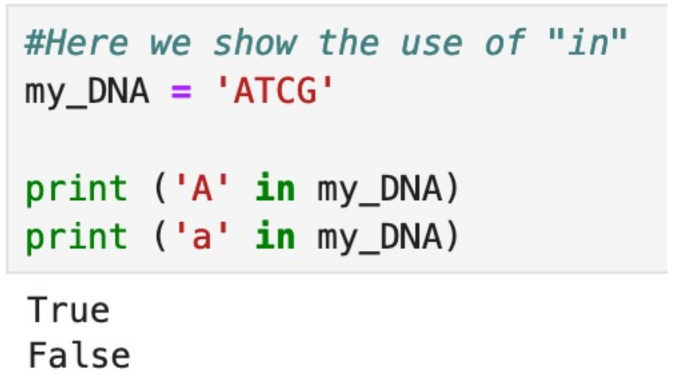
```python
#Here we show simple uses of the find method
my_DNA = 'ATCG'
print (my_DNA.find('A'))
print (my_DNA.find('T'))
print (my_DNA.find('a'))
```

```
0
1
-1
```

Text search

Evaluate, if substring is contained and count occurrences

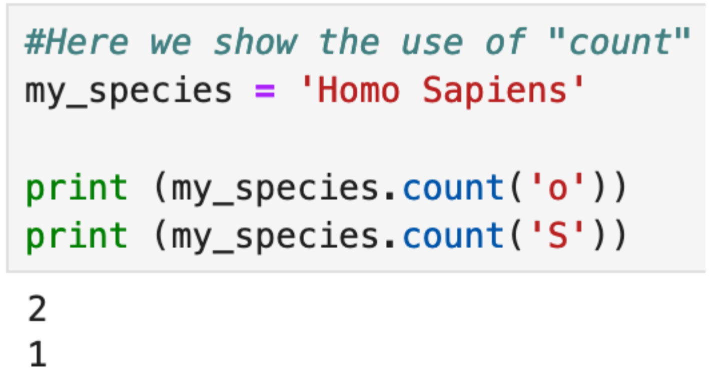
- If we are only interested in the information, if a substring is contained in a character string, we can the operator `in` in str context.



```
#Here we show the use of "in"
my_DNA = 'ATCG'

print ('A' in my_DNA)
print ('a' in my_DNA)
```

```
True
False
```

- We can count the occurrences of a substring using `str.count`



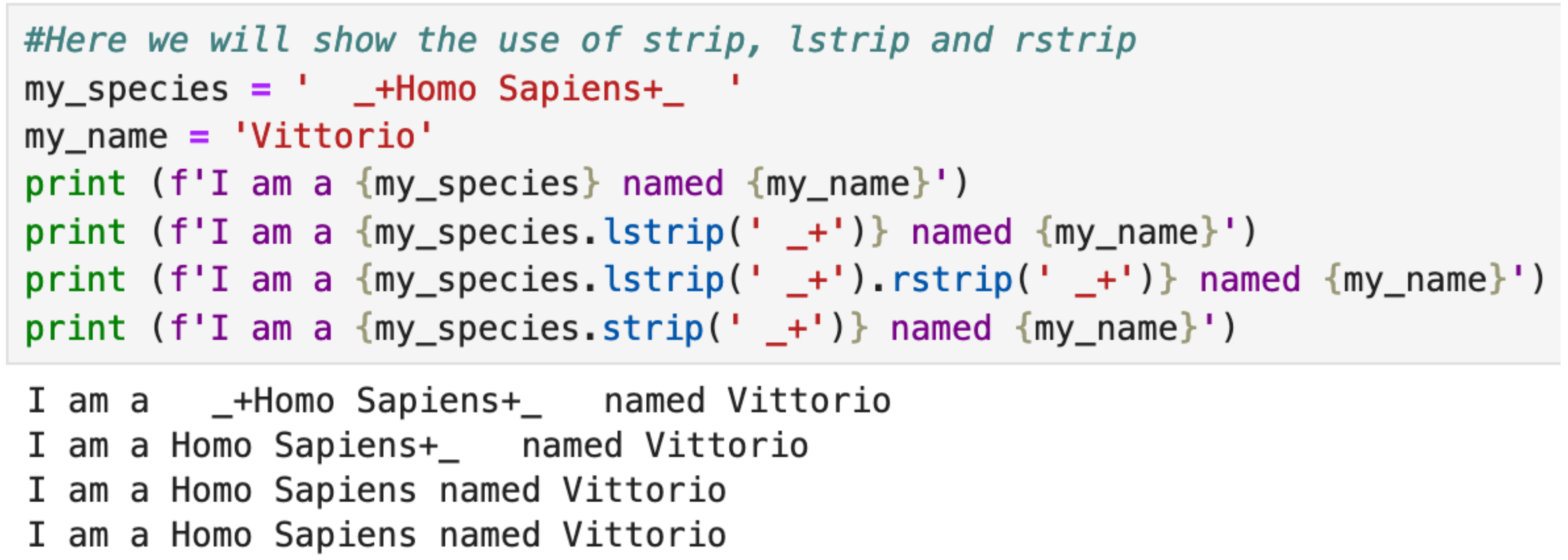
```
#Here we show the use of "count"
my_species = 'Homo Sapiens'

print (my_species.count('o'))
print (my_species.count('S'))
```

```
2
1
```

strip lstrip and rstrip

strip lstrip and rstrip are used to remove specific characters from a string

- The method `lstrip(…)` and `rstrip(…)` returns a copy of the character string in which any characters at the beginning (`lstrip`), end (`rstrip`) or both sides (`strip`) of the string are pruned (if present).
- The characters to be removed are given as `str` parameter.



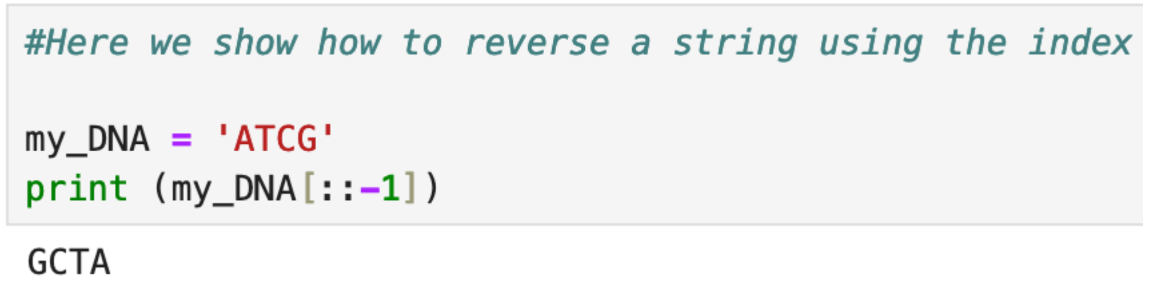
```python
#Here we will show the use of strip, lstrip and rstrip
my_species = '  _+Homo Sapiens+_  '
my_name = 'Vittorio'
print (f'I am a {my_species} named {my_name}')
print (f'I am a {my_species.lstrip(' _+')} named {my_name}')
print (f'I am a {my_species.lstrip(' _+').rstrip(' _+')} named {my_name}')
print (f'I am a {my_species.strip(' _+')} named {my_name}')
```

```
I am a   _+Homo Sapiens+_   named Vittorio
I am a Homo Sapiens+_   named Vittorio
I am a Homo Sapiens named Vittorio
I am a Homo Sapiens named Vittorio
```

Reversing a string

The pythonic way to reverse a string uses the index

- Objects with data type string are immutable
- Because character strings are immutable in Python, it is not possible to change a character at a certain position using indexing
- Any operation that alters the value stored in the object “creates” a new object with the same variable name and stores it
- The slicing operator `[start:stop]` can be used to access certain parts of a character string.
- To reverse a string with the index we use `str[::-1]`



```
#Here we show how to reverse a string using the index

my_DNA = 'ATCG'
print (my_DNA[::-1])
```

```
GCTA
```

CEPLAS
Cluster of Excellence on Plant Sciences

Introduction to Python

Break – 15 minutes

Conditional statements

`if`-statement

- We can use the `if`-statement to be able to distinguish between different states or conditions
- A specific action should only be taken, if a condition is fulfilled and evaluates as True
- The colon at the end of the line marks the beginning of a new block

```
if CONDITION:
    BLOCK (note the indentation, tab at the beginning of this line)
```

Block indentation

As already encountered when creating functions

- Python uses mandatory indentation of blocks of statements that are bound to a condition.
- All the lines of code that have to be executed if the condition in the conditional statement is met will follow the indentation.
- It is possible to substitute tabs with 4 consecutive space characters “ “ but they shouldn’t be mixed. As a general rule, choose one option and stick to it.
- The indentation is reduced when there are no more instructions that have to be executed if the conditional statement is fulfilled.

Conditional statements practical example

Traffic light example

- We will use pseudocode, a representation of code used to demonstrate the implementation of an algorithm without actually doing so.
- Let’s assume you are a pedestrian at a traffic light.

```
if traffic_ligt == red:
   wait
```

Control structures

`else`-statement

```
if CONDITION:
    BLOCK
else:
    BLOCK
```

- The `else`-statement can be omitted.
- Please note, that the else-syntax is not consisting of an explicit condition to be written.

Conditional statements practical example

Traffic light example

- Let’s assume you are a pedestrian at a traffic light.

```
if traffic_light == red:
    wait
else:
    check for cars left and right
    go
```

- Note that there is no condition associated with the else statement, if the condition is not met, instrutions in the else conditional block will be executed instead

Control structures

`elif`-statement

- Before the else-statement, it is possible to insert additional `elif` condition blocks. The statements in the `elif`-block will be executed, if preceding `if`- or `elif` conditions were False
- The else-block will be executed, if non of the preceding if- or elif-conditions were True.

```
if CONDITION:
    BLOCK
elif CONDITION:
    BLOCK
else:
    BLOCK
```

Conditional statement practical example

Traffic light example

- Let’s assume you are a pedestrian at a traffic light

```
if traffic_light == red:
   wait
elif traffic_light == orange:
   check for cars left and right
   check for police
   go fast!
else:
   check for cars left and right
   go
```

Conditional statements

Nested `if`-statements

- Within a conditional block, it is possible to have additional if statements
- This leads to nested blocks.

```
if traffic_light == red:
    wait
elif traffic_light == orange:
    if cars incoming == True
        wait
    elif police is looking at you == True
        wait
    else:
        go fast!
else:
    if cars incoming == True
        shout at them to make them stop
    go
```

Conditional statements

Boolean expressions

- A condition is every expression consisting of possible combinations of functions and operators, that can be evaluated as a Boolean (`True` or `False`)

- Numerical values are implicitly interpreted as Boolean. All numerical values not equal to 0 are interpreted as `True`, 0 is interpreted as `False`

- All non empty str objects are interpreted as `True`. Empty string objects are interpreted as `False`

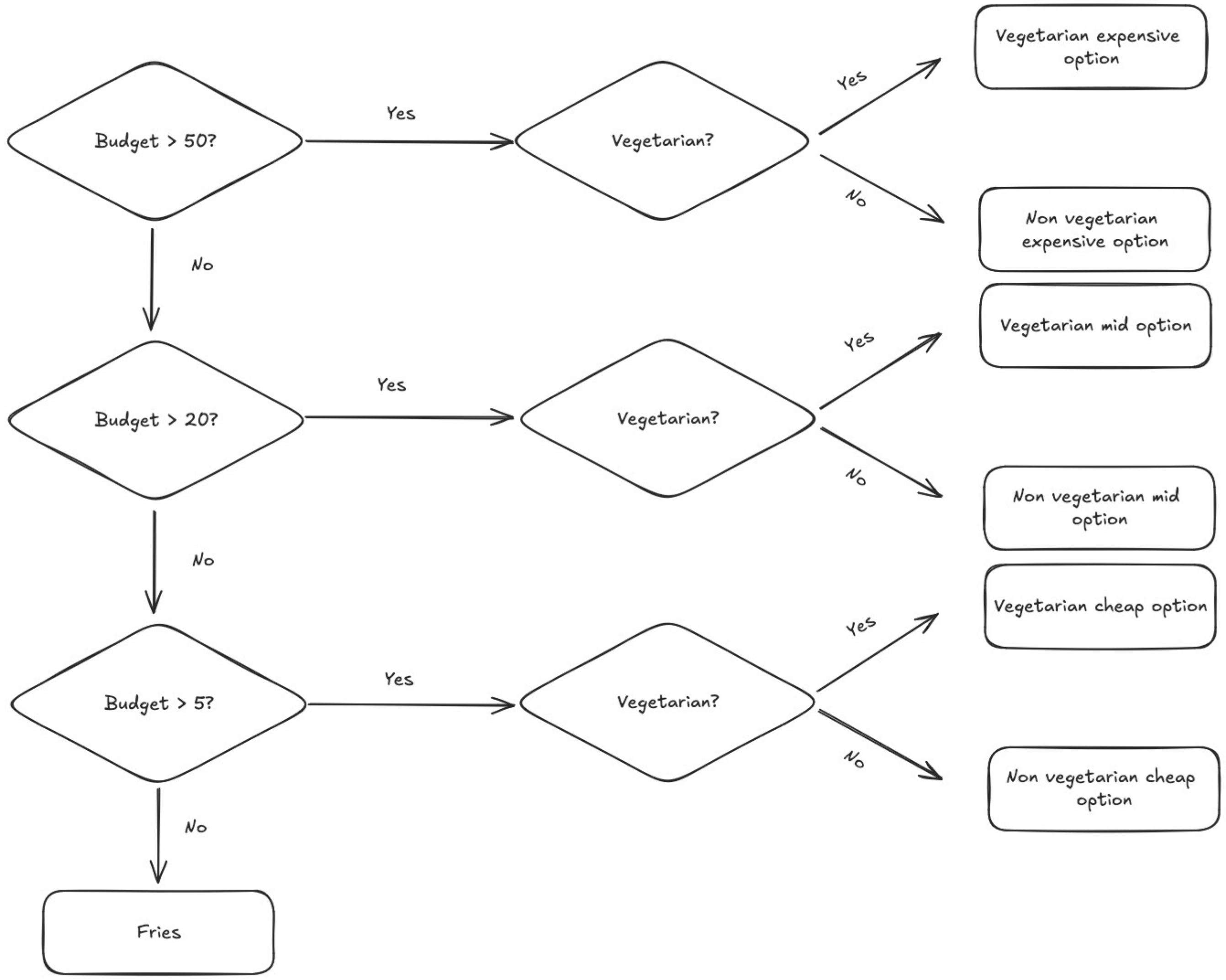
Hot question

Ordering at a restaurant

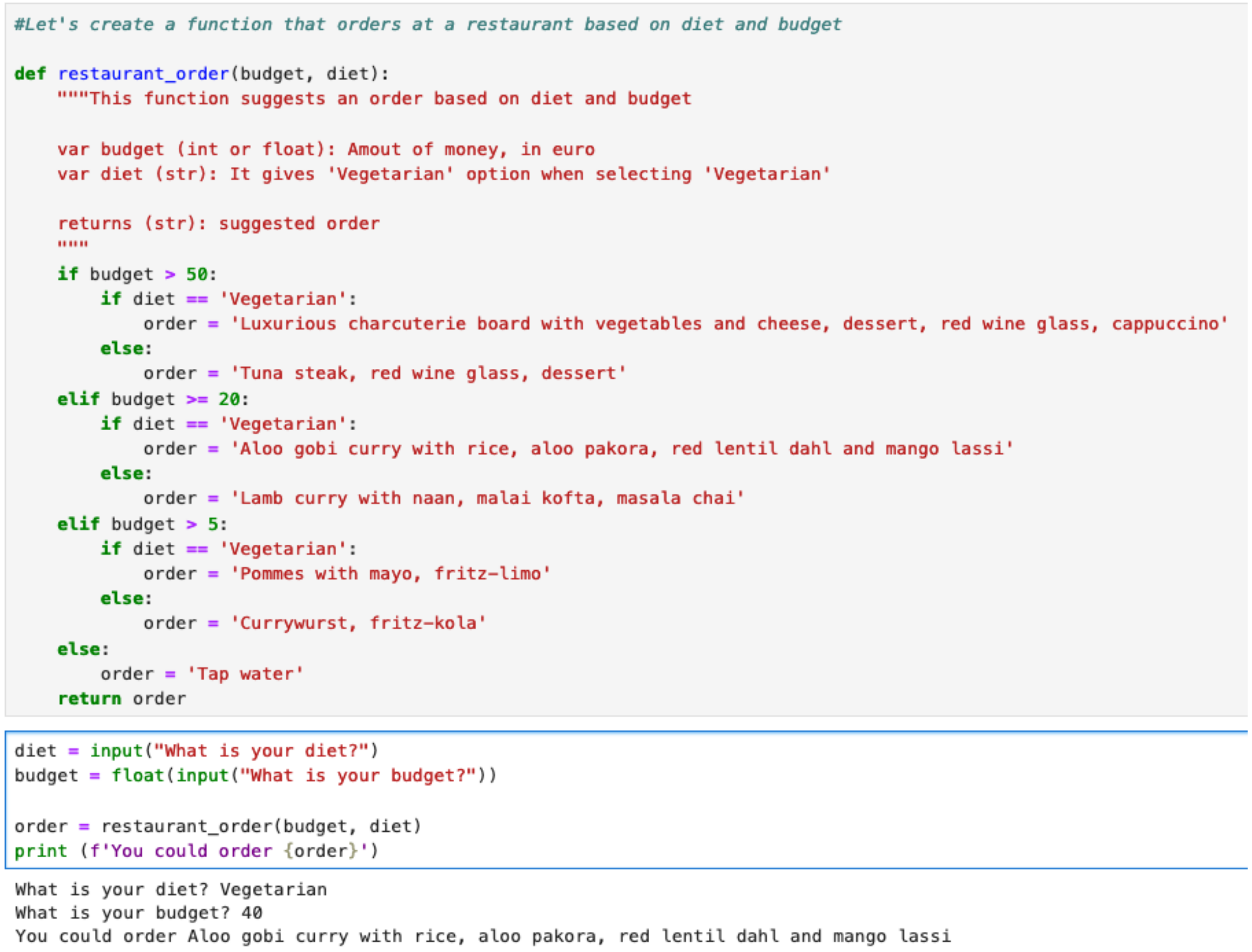
- Let’s create a script that suggests what to order at a brauhaus based on your budget and diet
- There are multiple ways to organize it. Try and minimize the number of conditions based on what we learned so far
- Use pseudocode, we will then look at an example using actual code

Pseudocode

What are the if, elif and else statements? How do you indent the blocks?



Hot question



```python
#Let's create a function that orders at a restaurant based on diet and budget

def restaurant_order(budget, diet):
    """This function suggests an order based on diet and budget

    var budget (int or float): Amout of money, in euro
    var diet (str): It gives 'Vegetarian' option when selecting 'Vegetarian'

    returns (str): suggested order
    """
    if budget > 50:
        if diet == 'Vegetarian':
            order = 'Luxurious charcuterie board with vegetables and cheese, dessert, red wine glass, cappuccino'
        else:
            order = 'Tuna steak, red wine glass, dessert'
    elif budget >= 20:
        if diet == 'Vegetarian':
            order = 'Aloo gobi curry with rice, aloo pakora, red lentil dahl and mango lassi'
        else:
            order = 'Lamb curry with naan, malai kofta, masala chai'
    elif budget > 5:
        if diet == 'Vegetarian':
            order = 'Pommes with mayo, fritz-limo'
        else:
            order = 'Currywurst, fritz-kola'
    else:
        order = 'Tap water'
    return order
```

```python
diet = input("What is your diet?")
budget = float(input("What is your budget?"))

order = restaurant_order(budget, diet)
print (f'You could order {order}')
```

```
What is your diet? Vegetarian
What is your budget? 40
You could order Aloo gobi curry with rice, aloo pakora, red lentil dahl and mango lassi
```

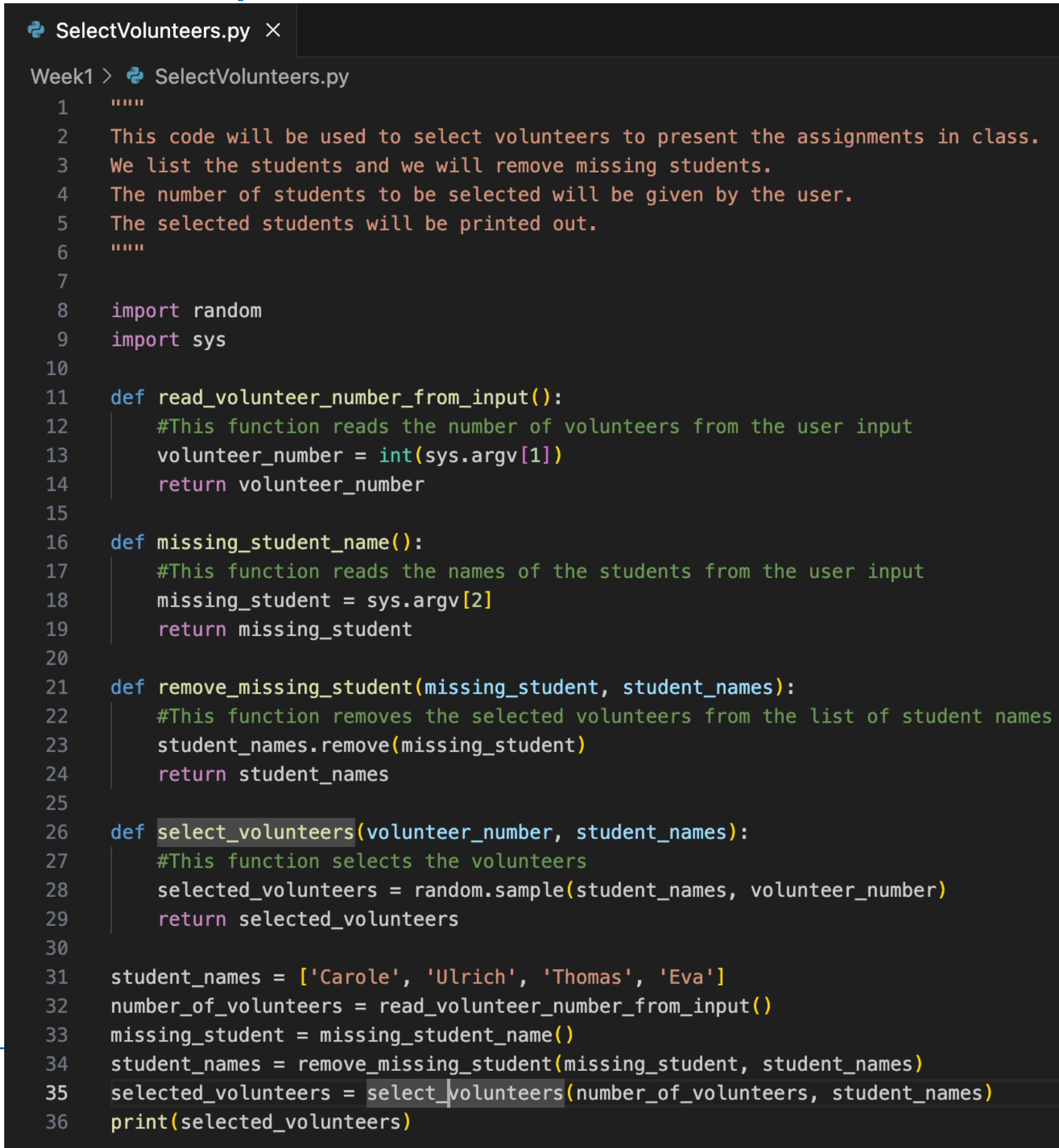
Improve selecting volunteers script – what we have so far

Let's use what we learned today to improve the script

- Use fstrings to improve the print statements

- Let's use upper and lower to make sure that the user input name matches our student list format (first letter capital, the rest low)

- Add a print statement that warns the user if the name selected is not a student in the class



SelectVolunteers.py

Week1 > SelectVolunteers.py

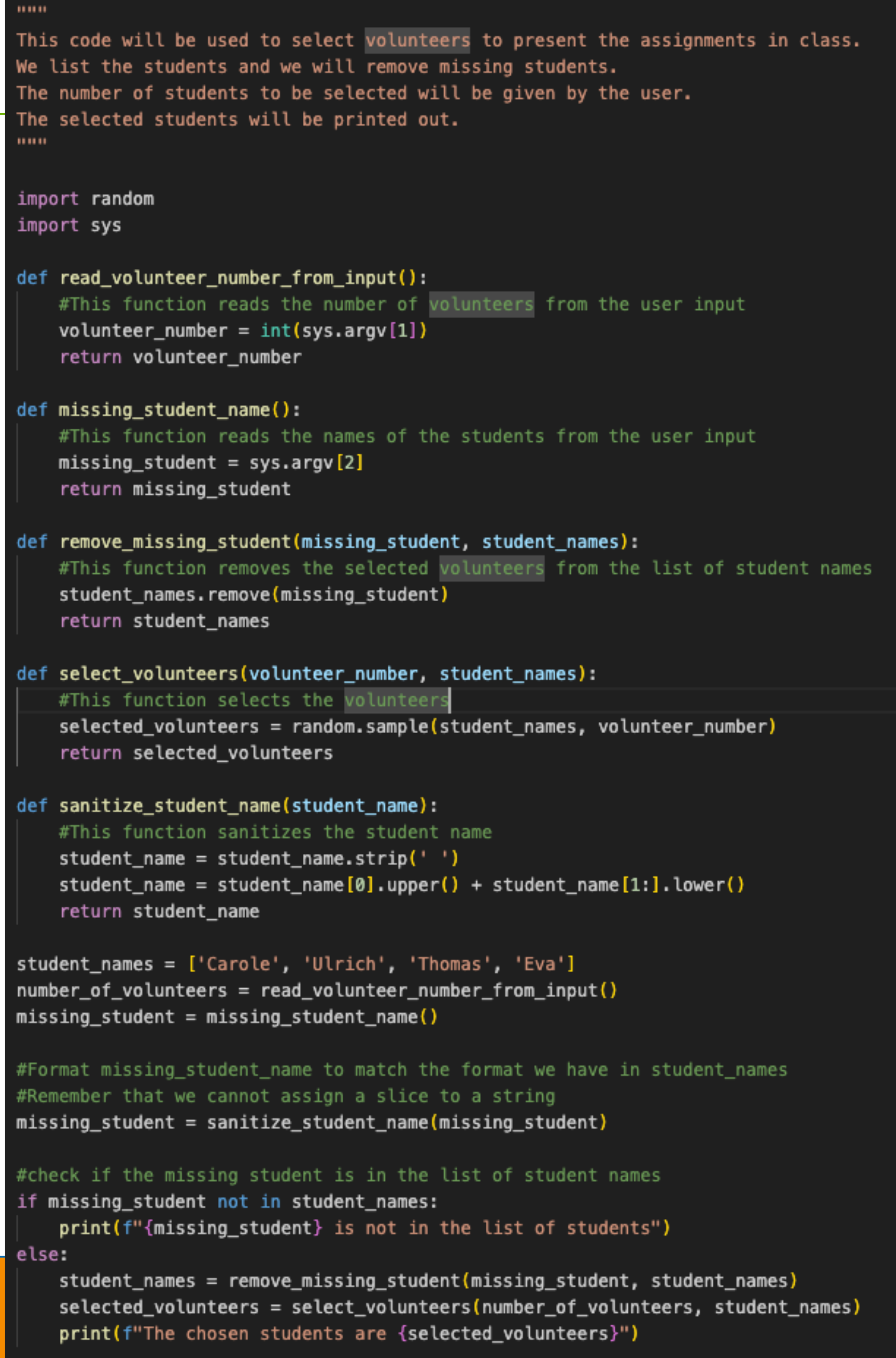
```python
"""
This code will be used to select volunteers to present the assignments in class.
We list the students and we will remove missing students.
The number of students to be selected will be given by the user.
The selected students will be printed out.
"""

import random
import sys

def read_volunteer_number_from_input():
    #This function reads the number of volunteers from the user input
    volunteer_number = int(sys.argv[1])
    return volunteer_number

def missing_student_name():
    #This function reads the names of the students from the user input
    missing_student = sys.argv[2]
    return missing_student

def remove_missing_student(missing_student, student_names):
    #This function removes the selected volunteers from the list of student names
    student_names.remove(missing_student)
    return student_names

def select_volunteers(volunteer_number, student_names):
    #This function selects the volunteers
    selected_volunteers = random.sample(student_names, volunteer_number)
    return selected_volunteers

student_names = ['Carole', 'Ulrich', 'Thomas', 'Eva']
number_of_volunteers = read_volunteer_number_from_input()
missing_student = missing_student_name()
student_names = remove_missing_student(missing_student, student_names)
selected_volunteers = select_volunteers(number_of_volunteers, student_names)
print(selected_volunteers)
```

Selecting volunteers

2.0

- Added formatted strings in print statements
- Added sanitize_student_name
- Handled the possible user mistake of incorrect capitalization of the names
- It now won’t accept names that are not in the list

- What do you think happened to the previous script if you wrote a name incorrectly?



```python
"""
This code will be used to select volunteers to present the assignments in class.
We list the students and we will remove missing students.
The number of students to be selected will be given by the user.
The selected students will be printed out.
"""

import random
import sys

def read_volunteer_number_from_input():
    #This function reads the number of volunteers from the user input
    volunteer_number = int(sys.argv[1])
    return volunteer_number

def missing_student_name():
    #This function reads the names of the students from the user input
    missing_student = sys.argv[2]
    return missing_student

def remove_missing_student(missing_student, student_names):
    #This function removes the selected volunteers from the list of student names
    student_names.remove(missing_student)
    return student_names

def select_volunteers(volunteer_number, student_names):
    #This function selects the volunteers
    selected_volunteers = random.sample(student_names, volunteer_number)
    return selected_volunteers

def sanitize_student_name(student_name):
    #This function sanitizes the student name
    student_name = student_name.strip(' ')
    student_name = student_name[0].upper() + student_name[1:].lower()
    return student_name

student_names = ['Carole', 'Ulrich', 'Thomas', 'Eva']
number_of_volunteers = read_volunteer_number_from_input()
missing_student = missing_student_name()

#Format missing_student_name to match the format we have in student_names
#Remember that we cannot assign a slice to a string
missing_student = sanitize_student_name(missing_student)

#check if the missing student is in the list of student names
if missing_student not in student_names:
    print(f"{missing_student} is not in the list of students")
else:
    student_names = remove_missing_student(missing_student, student_names)
    selected_volunteers = select_volunteers(number_of_volunteers, student_names)
    print(f"The chosen students are {selected_volunteers}")
```

CEPLAS
Cluster of Excellence on Plant Sciences

Question time & Recap

RECAP week 1

String formatting

Adding variables in print statements

Search, find, strip and reverse

Conditional statements

Nested conditional statements

Included new concepts in previous script