Python for Data Science - Exercises

Table of contents

Datatypes and Variables

Exercises 1

What does Python return as result of the following calculations?

```
print(f"10 + 10 + 10 = {10 + 10 + 10}")
print(f"10 - 3       = {10 - 3}")
print(f"10 * 2       = {10 * 2}")
print(f"5 / 2        = {5 / 2}")
print(f"5.0 / 2      = {5.0 / 2}")
print(f"5 % 2        = {5 % 2}")
print(f"10 % 3       = {10 % 3}")
print(f"3 ** 3       = {3 ** 3}")
print(f"9 ** 0.5     = {9 ** 0.5}")
print(f"10 + 3 * 3   = {10 + 3 * 3}")
print(f"(10 + 3) * 3 = {(10 + 3) * 3}")
```

```
10 + 10 + 10 = 30
10 - 3       = 7
10 * 2       = 20
5 / 2        = 2.5
5.0 / 2      = 2.5
5 % 2        = 1
10 % 3       = 1
3 ** 3       = 27
9 ** 0.5     = 3.0
10 + 3 * 3   = 19
(10 + 3) * 3 = 39
```

Try to find out: What happens?

```
a = 23
b = 23
print(f"a = {a}")
print(f"The ID of a is {id(a)}") # Prints memory address of the variable a
print(f"a is of type {type(a)}") # Prints the data type of the variable a
```

```
a = 23
The ID of a is 11761384
a is of type <class 'int'>
```

Note: `id()` corresponds to the objects memory address during its lifetime.

```
print(f"b = {b}")
print(f"The ID of b is {id(b)}")
print(f"b is of type {type(a)}")
```

```
b = 23
The ID of b is 11761384
b is of type <class 'int'>
```

Change the value of variable "a"

```
a = 123 # Reassign var a to int 123
print(f"a = {a}")
print(f"The ID of a is {id(a)}")
print(f"a is of type {type(a)}")
```

```
a = 123
The ID of a is 11764584
a is of type <class 'int'>
```

What happens now? why?

```
c = 23
print(f"c = {c}")
print(f"The ID of a is {id(c)}")
print(f"c is of type {type(c)}")
```

```
c = 23
The ID of a is 11761384
c is of type <class 'int'>
```

We assign the variable `c` to the value 23 and check its data type and memory location.

Try the same thing with a float variable!!!

```
d = 3.142
print(f"d = {d}")
print(f"The ID of a is {id(d)}")
print(f"d is of type {type(d)}")
```

```
d = 3.142
The ID of a is 130715404132112
d is of type <class 'float'>
```

Printing of text (strings)

```
print ("HSLU's Python course is cool")

print("This is Line 1")
print("This is Line 2")

print("\nWhat does the \\n control character do? Are you sure?") # \n is a line break

print("\tWhat does the \\t control character do? Are you sure?") # \t is a tabulator

print("I want to learn python because....")
print("\n\n\t....it's a cool data science programming language. Ok - there are also others ;-

print("What " + "is " + "wrong?" + "\nAdd some space between the words without removing the
```

```
HSLU's Python course is cool
This is Line 1
This is Line 2

What does the \n control character do? Are you sure?
```

```
    What does the \t control character do? Are you sure?
I want to learn python because....


    ....it's a cool data science programming language. Ok - there are also others ;-) !
What is wrong?
Add some space between the words without removing the '+' signs!
```

try to understand the behaviour

```python
g=h='hslu'      # Assign the var g and h to the string "hslu"
print(g)        # Prints hslu
print(h)        # Print hslu
h=456           # Reassign h to int 456
print(h)        # Prints 456
h=h+1           # Calculates 456 + 1 = 457
print(h)        # Prints 457
h+=1            # Calculates 457 + 1 = 458
print(h)        # Prints 458
p='hslu'        # Assign the var p to the string "hslu"
q=p             # Copies the var p (hlsu) to the var q
print(p==q)     # Compare the var p and q (Output: True)
print(p is q)   # Compare the var p and q (Output: True)
```

```
hslu
hslu
456
457
458
True
True
```

Conditions

Task 1

Create a simple program in which three whole numbers (potentially negative!) are read in from the keyboard and afterwards get displayed on the screen in ascending order. Use nested if-statements for order determination (i.e. no built-in- 'sort' function).

Using if-else-statements

```
inputs = [] # Set up storage

print("Please input 3 whole numbers of your choise.")
"""
input1 = int(input("Number one: ")) # Assign input to a variable
input2 = int(input("Number two: "))
input3 = int(input("Number tree: "))
"""

# Use fake user inputs
input1 = -3
input2 = 10
input3 = -6

# Sort list
if input1 < input2 and input1 < input3:
    inputs.append(input1) # Append input to list
    if input2 < input3:
        inputs.append(input2)
        inputs.append(input3)
    else:
        inputs.append(input3)
        inputs.append(input2)
elif input2 < input1 and input2 < input3:
    inputs.append(input2)
    if input1 < input3:
        inputs.append(input1)
        inputs.append(input3)
    else:
        inputs.append(input3)
        inputs.append(input1)
elif input3 < input1 and input3 < input2:
    inputs.append(input3)
    if input1 < input2:
        inputs.append(input1)
        inputs.append(input2)
    else:
        inputs.append(input2)
        inputs.append(input1)
```

```
print(inputs)
```

```
Please input 3 whole numbers of your choise.
[-6, -3, 10]
```

Using build in function `sorted()`

```
inputs = [] # Set up storage

print("Please input 3 whole numbers of your choise.")
"""
input1 = int(input("Number one: ")) # Assign input to a variable
inputs.append(input1)
input2 = int(input("Number two: "))
inputs.append(input2)
input3 = int(input("Number tree: "))
inputs.append(input3)
"""

# Use fake user inputs
input1 = -3
inputs.append(input1)
input2 = 10
inputs.append(input2)
input3 = -6
inputs.append(input3)

# Sort list
inputs = sorted(inputs)
print(inputs)
```

```
Please input 3 whole numbers of your choise.
[-6, -3, 10]
```

Task 2

a

Assumption: The int variables a, b and c are declared and initialized.

Define a conditional expression “EXPR” in such a way that the body of the expression

if EXPR: print(“condition fulfilled”)

is only carried out if: a is greater than b OR a is less than half of b OR the sum of a and c is greater than b.

Check your result: a=1, b=2, c=2 -> True a=1, b=2, c=1 -> False

```
# Test case one
a = 1
b = 2
c = 2

if a > b or a < b/2 or sum([a, c]) > b:
    print("condition fulfilled")
else:
    print("condition not fulfilled")
```

```
condition fulfilled
```

```
# Test case two
a = 1
b = 2
c = 1

if a > b or a < b/2 or sum([a, c]) > b:
    print("condition fulfilled")
else:
    print("condition not fulfilled")
```

```
condition not fulfilled
```

b

Assumption: The int variables a, b and c are declared and initialized.

Define a conditional expression “EXPR” in such a way that the body of the expression

if EXPR: print(“condition fulfilled”)

is only carried out if: half of the number a is an odd number OR the subtraction of the numbers b and c results in an even number OR both a and b and also b and c have different values.

Check your result: a=6, b=2, c=0 => True a=5, b=2, c=1 => True a=5, b=5, c=2 => False

```
# Test case one
a = 6
b = 2
c = 0

if a/2 % 2 == 1 or b - c % 2 == 0 or a != b and b != c:
        print("condition fulfilled")
else:
    print("condition not fulfilled")
```

```
condition fulfilled
```

```
# Test case two
# Test case one
a = 5
b = 2
c = 1

if a/2 % 2 == 1 or b - c % 2 == 0 or a != b and b != c:
        print("condition fulfilled")
else:
    print("condition not fulfilled")
```

```
condition fulfilled
```

```
# Test case three
# Test case one
a = 5
b = 5
c = 2

if a/2 % 2 == 1 or b - c % 2 == 0 or a != b and b != c:
        print("condition fulfilled")
else:
    print("condition not fulfilled")
```

```
condition not fulfilled
```

Loops and Conditions

Exercise 1

Write a python program that constructs the following pattern using a nested for loop:

```
*
**
***
****
*****
****
***
**
*
```

```
upper  = range(0, 6, 1) # Define upper pattern
lower = range(6, 0, -1) # Define lower pattern


for i in upper:
    print("*" * i) # Print upper part of pattern
else: # After done
    for j in lower:
        print("*" * j) # Print lower part of pattern
```

```
*
**
***
****
*****
******
*****
****
***
**
*
```

Exercise 2

Write a python program to count the number of even and odd numbers contained in a sequence of numbers.

```
even_numbers = []
odd_numbers = []
numbers = (1,2,3,4,5,6,7,8,9,10)

for i in range(len(numbers)):
    if i % 2 == 0: # Check if current number is even
        even_numbers.append(i)
    else:
        odd_numbers.append(i)

print(f"Total number of even numbers: {len(even_numbers)}") # Use len() to count
print(f"Total number of even numbers: {len(odd_numbers)}")
```

```
Total number of even numbers: 5
Total number of even numbers: 5
```

Exercise 3

Write a Python program which takes two digits m (row) and n (column) as input and generates a two-dimensional structure (e.g. nested list). The value in the i-th row and j-th column should be i*j.

```
rows = 3
columns = 4
result = [] # Define master list

for row in range(rows):
    sub_list = [] # Define sub-list
    for column in range(columns):
        sub_list.append(row * column) # Calc value
    result.append(sub_list) # Append sub-list to master list

print(result)
```

```
[[0, 0, 0, 0], [0, 1, 2, 3], [0, 2, 4, 6]]
```

Exercise 4

Write a Python program to find all numbers between 200 and 500 (limits included) only containing even digits. Correct solution:

```python
my_range = range(200, 501) # Include boarders
even_digit_numbers = [] # Final result

for number in my_range:
    test_result = []
    for digit in str(number):
        if int(digit) % 2 == 1:
            test_result.append(True) # Add true for odd digit
        else:
            test_result.append(False) # Add false for even digit
    if any(test_result):
        pass # Pass if any odd digit (false) is in test_result
    else:
        even_digit_numbers.append(number) # Append to final result

print(even_digit_numbers)
```

```
[200, 202, 204, 206, 208, 220, 222, 224, 226, 228, 240, 242, 244, 246, 248, 260, 262, 264, 2
```

Exercise 5

Write a Python program to guess a number between 1 to 10. First: Use the predefined code block to create a random number Second: The user is prompted to enter a guess. If the guess is wrong a message "to big" or "to small" is printed to the console and the prompt (user input) appears again until the guess is correct. If the guess is correct, "Well guessed!" will be printed and the program ends. Extension: The number of trials should be prompted as well: "Well done - you have tried it 4 times!"

```python
# Code block to create a random number
from random import randint
random_number = randint(1,10)
# End code block to create a random number

trials = 0

while True: # Infinitive loop till break
    # Unsing fake user input
    user_input = randint(1,10) # Also random
    trials += 1 # Count trials
    if user_input == random_number:
        print(f"Well done - you have tried it {trials} times!")
```

```
        break
    elif user_input < random_number:
        print("Guess to low...")
    else:
        print("Guess to high...")
```

```
Guess to high...
Well done - you have tried it 2 times!
```

Exercise 6

Suppose we wish to draw a Christmas tree. Example tree - desired height: 8 The result looks like:

```
     x
    xxx
   xxxxx
  xxxxxxx
 xxxxxxxxx
xxxxxxxxxxx
    xxx
    xxx
```

Create a program that asks the user for the height of the Christmas tree (trunk included) and then draws the tree. The height of the trunk is always 2 and the width is 3 in case the entire height is > 5 and 1 in case the tree is smaller.

```
total_height = 8
trunk_height = 2
crown_height = total_height - trunk_height
max_crown_width = 2 * (crown_height - 1) + 1

if total_height > 5:
    trunk_width = 3
else:
    trunk_width = 1

i = 0
while i < crown_height:
    current_crown_width = 2 * i + 1
    spaces = (max_crown_width - current_crown_width) // 2
```

```
    print(" " * spaces + "x" * current_crown_width)
    i += 1

trunk_padding = (max_crown_width - trunk_width) // 2

j = 0
while j < trunk_height:
    print(" " * trunk_padding + "x" * trunk_width)
    j += 1
```

```
     x
    xxx
   xxxxx
  xxxxxxx
 xxxxxxxxx
xxxxxxxxxxx
    xxx
    xxx
```

Debug

Exercise 1

Write a Python program to guess a number between 1 to 10. First: Use the predefined code block to create a random number Second: The user is prompted to enter a guess. If the guess is wrong a message "to big" or "to small" is printed to the console and the prompt (user input) appears again until the guess is correct. If the guess is correct, "Well guessed!" will be printed and the program ends. Extension: The number of trials should be prompted as well: "Well done - you have tried it 4 times!"

```
# Code block to create a random number
from random import randint
random_number = randint(1,10)
# End code block to create a random number

trials = 0

while True: # Infinitive loop till break
    # Unsing fake user input
    user_input = randint(1,10) # Also random
```

```
    trials += 1 # Count trials
    if user_input == random_number:
        print(f"Well done - you have tried it {trials} times!")
        break
    elif user_input < random_number:
        print("Guess to low...")
    else:
        print("Guess to high...")
```

```
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to high...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to high...
Guess to low...
Guess to high...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
Guess to low...
```

```
Guess to low...
Guess to low...
Well done - you have tried it 36 times!
```

Exercise Collection Functions

Exercise 1

Write a function that accepts a list of integers as input and returns a list containing all even numbers of the input list.

```python
def extract_even_num(num_list: list[int]) -> list:
    """
    Returns all even numbers of within a given list

    Args:
        num_list: A list with integers.

    Returns:
        list: A list with all even numbers from num_list
    """
    even_num_list = [] # Result storage
    for num in num_list:
        if num % 2 == 0: # Check for even or odd
            even_num_list.append(num)

    return even_num_list # Return result

even_num = extract_even_num((1, 2, 3 ,4, 5, 6, 7, 8, 9))
print(even_num)
```

```
[2, 4, 6, 8]
```

Exercice 2

Write a function accepts a list as input and returns the unique values contained in that list as a list. Hint: you can use the “set” method to check your solution.

```
def get_unique_values(value_list: list) -> set:
    """
    Return a set (unique values) from a given list

    Args:
        value_list: List with different values

    Returns:
        set: Unique values from value_list
    """
    return set(value_list)

uniques = get_unique_values((1, 2, 2, 3, "Hello", "Hello", "Bye"))
print(uniques)
```

```
{1, 2, 3, 'Hello', 'Bye'}
```

Exercise 3

Crate a simple calculator that provides functions to add, subtract, multiply and divide two numbers. Implement the missing parts to execute the main() function below without an error.

```
def add(num1: int, num2: int) -> int:
    """ Add two numbers """
    return num1 + num2

def subtract(num1: int, num2: int) -> int:
    """ Subtract two numbers """
    return num1 - num2

def multiply(num1: int, num2: int) -> int:
    """ Multiply two numbers """
    return num1 * num2

def divide(num1: int, num2: int) -> int:
    """ Divide two numbers """
    return num1 / num2

def main():
    print("Select operation.")
```

```
    print("1.Add")
    print("2.Subtract")
    print("3.Multiply")
    print("4.Divide")

    # Take input from the user
    # choice = input("Enter choice(1/2/3/4):")
    # Using fake user inputs
    choice = str(1)

    # num1 = int(input("Enter first number: "))
    # num2 = int(input("Enter second number: "))
    # Using fake user inputs

    num1 = 4
    num2 = 5

    if choice == '1':
       print(num1,"+",num2,"=", add(num1,num2))

    elif choice == '2':
       print(num1,"-",num2,"=", subtract(num1,num2))

    elif choice == '3':
       print(num1,"*",num2,"=", multiply(num1,num2))

    elif choice == '4':
       print(num1,"/",num2,"=", divide(num1,num2))
    else:
       print("Invalid input")

main()
```

```
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
4 + 5 = 9
```

Extercise 4

Keyword parameters If we want call the function circumference(2) with the 'width' 2 we have a problem: The number '2' is by default assigned to the first Parameter and that is 'length' ... Task: Change the calls so that the function circumference(2) is called with 'width' 2!

```
def circumference(length = 2, width = 1):
     return 2 * (length + width)

c1 = circumference(width=2)         # 2 should be the width !!!
print(c1)

# or with length 5 and width 3:
c2 = circumference(5, 3)
print(c2)

# how could the function also have been called
# with length 5 and width 3
c3 = circumference(length=5, width=3)
print(c3)

# you can even change the order of keyword parameters
# try it: Change the order of  parameters!
c4 = circumference (width=3, length=5)     # order swapped!
print(c4)
```

```
8
16
16
16
```

Exercise 5

Functions may have parameters which may have default values.

Task: Change the two examples (1 and 2) such that the two function calls hello(...) and the three function calls circumference(...) run without any error.

```
# Example 1:
def hello (name = "Mr. Unknown"):
     print ("Hello " + name + "!")
```

```
hello("Peter")

# without parameters (default value used)
hello()


# Example 2:
def circumference (length=1, width=1):
     return 2 * (length + width)

c1 = circumference(5, 3)
print(c1)

# for the width, the default value "1" is used.
c2 = circumference(5)
print(c2)

# both default values are used.
c3 = circumference()
print(c3)
```

```
Hello Peter!
Hello Mr. Unknown!
16
12
4
```

Exercise Collection File Access

Exercise 01

Write a program that combines three lines from the file 'musicians.txt' into one line and writes them as standard output using print(). That means: An output line should contain: first name, last name and artist name ("Künstername") or band of the artist. The band or artist name should be printed in round brackets and in captial letters. Example: "Marshall Bruce Mathers III (EMINEM)"

```
with open('data/musicians.txt', mode='r') as file: # Read file
        musicians = [line.rstrip() for line in file]
```

```
first_name = 0
artist_name = 1
band = 2

while band < len(musicians):
    print(f"{musicians[first_name]} {musicians[artist_name]} ({str.upper(musicians[band])})")

    first_name += 3
    artist_name += 3
    band += 3
```

```
Brian Molko (PLACEBO)
Jim Morrison (THE DOORS)
Ray Davies (THE KINKS)
Marshall Bruce Mathers III (EMINEM)
Andre Romelle Young (DR. DRE)
Beth Gibbons (PORTISHEAD)
```

Exercise 02

Change the solution of Exercise 02 so that the program writes the output to the file 'musicians2.txt'.

```
with open('data/musicians.txt', mode='r') as file: # Read file
        musicians = [line.rstrip() for line in file]

first_name = 0
artist_name = 1
band = 2

while band < len(musicians):
    with open('data/musicians2.txt', mode='a') as file2: # Mode = Append
        file2.write(f"{musicians[first_name]} {musicians[artist_name]} ({str.upper(musicians

    first_name += 3
    artist_name += 3
    band += 3
```

Exercise Collection String Formatting

Exercise 01

Use the f"XXX" syntax to produce the following print out "Hello Andreas and Ramon!" based on the given varibles.

```
var1 = "and"
var2 = "Andreas"
var3 = "!"
var4 = "Hello"
var5 = "Ramon"

print(f"{var4} {var2} {var1} {var5}{var3}")
```

```
Hello Andreas and Ramon!
```

Exercise 02

Use the "XXX".format() syntax to produce the following print out: "My name is Giacomo, I am 10 years old, and I live in Padova." based on the given dictionary. In the formatting call use: 1 order based 2 numbering based 3 key word based 4 dictionary key based versions to insert the variables.

```
insert_dict = {'name': 'Giacomo', 'age': 10, 'city': 'Padova'}

# Order based
print("My name is {}, I am {} years old, and I live in {}.".format(
    insert_dict['name'], insert_dict['age'], insert_dict['city']
))

# Numbering based
print("My name is {0}, I am {1} years old, and I live in {2}.".format(
    insert_dict['name'], insert_dict['age'], insert_dict['city']
))

# Key word based
print("My name is {n}, I am {a} years old, and I live in {c}.".format(
    n=insert_dict['name'], a=insert_dict['age'], c=insert_dict['city']
))
```

```python
# Dictionary key based
print("My name is {name}, I am {age} years old, and I live in {city}.".format(
    **insert_dict
))
```

```
My name is Giacomo, I am 10 years old, and I live in Padova.
My name is Giacomo, I am 10 years old, and I live in Padova.
My name is Giacomo, I am 10 years old, and I live in Padova.
My name is Giacomo, I am 10 years old, and I live in Padova.
```

Exercise 03

You are given list of n (even) numbers. Write a program that prints for each pair [(first, last), (second, second last), etc…] X and Y the following string: X times Y is: X*Y. Use in place calculations to calculate the product directly in the string within the print statement.

```python
nums = [1, 2, 3, 4, 5, 6]

for i in range(len(nums)):
    print(f"{nums[i] * nums[len(nums) - 1 - i]}")
```

```
6
10
12
12
10
6
```

Exercise kwargs

Write a function that accepts an arbitrary number of keyword arguments. The function should number each keyword-value pair and print it in the following format: 1: keyword -> value 2: keyword -> value … n: keyword -> value

```python
def print_pairs(**kwargs): # Takes n parameters
    # Iterate over dict
    for i, (key, value) in enumerate(kwargs.items(), start=1):
        print(f"{i}: {key} -> {value}")

print_pairs(first=1, second=2, third=3)
```

```
1: first -> 1
2: second -> 2
3: third -> 3
```

Exercise Animals

Part a

For each animal, print “Hi, I’m a {animal_type} and my name is {name}!” Solve the exercise by writing a class ‘Animal’ and adding a method allowing to print the requested string. Note that it is not necessary for this task, to differentiate between animal types!

```
print("Part a): \n\n")

class Animal:

    def __init__(self, animal_type, name, age, weight, legs):
        self.animal_type = animal_type
        self.name = name
        self.age = age
        self.weight = weight
        self.legs = legs

    def __str__(self):
        return(f"Hi, I'm a {self.animal_type} and my name is {self.name}!")

with open('data/animals.csv') as f:
    for line in f.readlines()[1:]:
        animal_type = line.split(", ")[0]
        name = line.split(", ")[1]
        age = int(line.split(", ")[2])
        weight = int(line.split(", ")[3])
        legs = int(line.split(", ")[4])
        #create objects
        obj = Animal(animal_type, name, age, weight, legs)
        print(obj)
```

```
Part a):


Hi, I'm a dog and my name is bello!
```

```
Hi, I'm a snake and my name is masha!
Hi, I'm a dog and my name is wuffi!
Hi, I'm a cat and my name is leslie!
Hi, I'm a cat and my name is dummy!
Hi, I'm a cat and my name is puffy!
Hi, I'm a snake and my name is snizzy!
Hi, I'm a snake and my name is sazzles!
Hi, I'm a dog and my name is buddy!
Hi, I'm a cat and my name is wooly!
Hi, I'm a cat and my name is beast!
```

Part b

Now we are interested in the various sounds the animal types make. Write a separate class for each animal type. Provide a method "speak", which depending on the animal type prints different sounds.

```python
print("\n\nPart b): \n\n")

class Dog():

    def __init__(self, name, age, weight, legs):
        self.name = name
        self.age = age
        self.weight = weight
        self.legs = legs

    def speak(self):
        return "Whuff whuff!"

class Cat():

    def __init__(self, name, age, weight, legs):
        self.name = name
        self.age = age
        self.weight = weight
        self.legs = legs

    def speak(self):
        return "Mew mew!"

class Snake():
```

```
    def __init__(self, name, age, weight, legs):
        self.name = name
        self.age = age
        self.weight = weight
        self.legs = legs

    def speak(self):
        return "Sssssss!"


# solution with file loop
obj_list = []
with open('data/animals.csv') as f:
    for line in f.readlines()[1:]:
        animal_type = line.split(", ")[0]
        name = line.split(", ")[1]
        age = int(line.split(", ")[2])
        weight = int(line.split(", ")[3])
        legs = int(line.split(", ")[4])
        #create objects
        if animal_type == 'dog':
            obj = Dog(name, age, weight, legs)
            obj_list.append(obj)
        if animal_type == 'cat':
            obj = Cat(name, age, weight, legs)
            obj_list.append(obj)
        if animal_type == 'snake':
            obj = Snake(name, age, weight, legs)
            obj_list.append(obj)

for elem in obj_list:
    print(elem.speak())
```

Part b):

```
Whuff whuff!
Sssssss!
Whuff whuff!
Mew mew!
```

```
Mew mew!
Mew mew!
Sssssss!
Sssssss!
Whuff whuff!
Mew mew!
Mew mew!
```

Exercise Animals Inheritance

Part a

Produce a class structure where Dog, Cat and Snake are implemented as child classes of a base class Animal. Create the object variables in the base class, along with a method 'print_basic_information', which prints 'type' and 'name' to the console. In each child class, use the base class constructor with 'super' and add an additional method 'speak', specific to that child class. In each child class, add an additional method where you return the sum of the object variables 'age' and 'legs'.

For each element in the animals.csv, create an object of the corresponding subclass, let it print basic information (method from 'Animal'), let it speak (method from child class) and get the sum of 'age' and 'legs' as a return value.

```python
class Animal:
    def __init__(self, animal_type: str, name: str, age: int, weight: int, legs: int):
        self.animal_type = animal_type
        self.name = name
        self.age = age
        self.weight = weight
        self.legs = legs

    def print_basic_information(self):
        print(f"Type = {self.animal_type} and name = {self.name}")

class Dog(Animal):

    def __init__(self, animal_type: str, name: str, age: int, weight: int, legs: int):
        super().__init__(animal_type, name, age, weight, legs)

    def speak(self):
        return "Wuff"
```

```
    def sum_of_age_and_legs(self):
        return sum([self.age, self.legs])

class Cat(Animal):

    def __init__(self, animal_type: str, name: str, age: int, weight: int, legs: int):
        super().__init__(animal_type, name, age, weight, legs)

    def speak(self):
        return "Meww"

    def sum_of_age_and_legs(self):
        return sum([self.age, self.legs])

class Snake(Animal):

    def __init__(self, animal_type: str, name: str, age: int, weight: int, legs: int):
        super().__init__(animal_type, name, age, weight, legs)

    def speak(self):
        return "Ssss"

    def sum_of_age_and_legs(self):
        return sum([self.age, self.legs])

obj_list = []
with open('./data/animals.csv') as f:
    for line in f.readlines()[1:]:
        animal_type = line.split(", ")[0]
        name = line.split(", ")[1]
        age = int(line.split(", ")[2])
        weight = int(line.split(", ")[3])
        legs = int(line.split(", ")[4])
        # create objects based on animal_type
        if animal_type == "dog":
            obj = Dog(animal_type=animal_type, name=name, age=age, weight=weight, legs=legs)
        elif animal_type == "cat":
            obj = Cat(animal_type=animal_type, name=name, age=age, weight=weight, legs=legs)
        else:
            obj = Snake(animal_type=animal_type, name=name, age=age, weight=weight, legs=legs
        # add objects to obj_list
        obj_list.append(obj)
```

```
for elem in obj_list:
    print("--\n")
    elem.print_basic_information()
    elem.speak()
    c_sum = elem.sum_of_age_and_legs()
    print(c_sum)
```

```
--

Type = dog and name = bello
14
--

Type = snake and name = masha
12
--

Type = dog and name = wuffi
12
--

Type = cat and name = leslie
9
--

Type = cat and name = dummy
9
--

Type = cat and name = puffy
9
--

Type = snake and name = snizzy
5
--

Type = snake and name = sazzles
2
--
```

```
Type = dog and name = buddy
6
--

Type = cat and name = wooly
9
--

Type = cat and name = beast
9
```

Part b

```python
class Animal:
    def __init__(self, animal_type: str, name: str, age: int, weight: int, legs: int, sound:
        self.animal_type = animal_type
        self.name = name
        self.age = age
        self.weight = weight
        self.legs = legs
        self.sound = sound

    def print_basic_information(self):
        print(f"Type = {self.animal_type} and name = {self.name}")

    def sum_of_age_and_legs(self):
        return sum([self.age, self.legs])

    def speak(self):
        print(self.sound)

class Dog(Animal):

    def __init__(self, animal_type: str, name: str, age: int, weight: int, legs: int, sound:
        super().__init__(animal_type, name, age, weight, legs, sound)

class Cat(Animal):

    def __init__(self, animal_type: str, name: str, age: int, weight: int, legs: int, sound:
        super().__init__(animal_type, name, age, weight, legs, sound)
```

```
class Snake(Animal):

    def __init__(self, animal_type: str, name: str, age: int, weight: int, legs: int, sound:
        super().__init__(animal_type, name, age, weight, legs, sound)

obj_list = []
with open('./data/animals.csv') as f:
    for line in f.readlines()[1:]:
        animal_type = line.split(", ")[0]
        name = line.split(", ")[1]
        age = int(line.split(", ")[2])
        weight = int(line.split(", ")[3])
        legs = int(line.split(", ")[4])
        # create objects based on animal_type
        if animal_type == "dog":
            obj = Dog(animal_type=animal_type, name=name, age=age, weight=weight, legs=legs,
        elif animal_type == "cat":
            obj = Cat(animal_type=animal_type, name=name, age=age, weight=weight, legs=legs,
        else:
            obj = Snake(animal_type=animal_type, name=name, age=age, weight=weight, legs=legs
        # add objects to obj_list
        obj_list.append(obj)

for elem in obj_list:
    print("--\n")
    elem.print_basic_information()
    elem.speak()
    c_sum = elem.sum_of_age_and_legs()
    print(c_sum)
```

```
--

Type = dog and name = bello
Wuff
14
--

Type = snake and name = masha
Ssss
12
--
```

```
Type = dog and name = wuffi
Wuff
12
--

Type = cat and name = leslie
Meww
9
--

Type = cat and name = dummy
Meww
9
--

Type = cat and name = puffy
Meww
9
--

Type = snake and name = snizzy
Ssss
5
--

Type = snake and name = sazzles
Ssss
2
--

Type = dog and name = buddy
Wuff
6
--

Type = cat and name = wooly
Meww
9
--

Type = cat and name = beast
Meww
9
```

Part c

```
class Animal:
    def __init__(self, animal_type: str, name: str, age: int, weight: int, legs: int, sound:
        self.animal_type = animal_type
        self.name = name
        self.age = age
        self.weight = weight
        self.legs = legs
        self.sound = sound

    def print_basic_information(self):
        print(f"Type = {self.animal_type} and name = {self.name}")

    def sum_of_age_and_legs(self):
        return sum([self.age, self.legs])

    def speak(self):
        print(self.sound)

class Dog(Animal):

    def __init__(self, animal_type: str, name: str, age: int, weight: int, legs: int, sound:
        super().__init__(animal_type, name, age, weight, legs, sound)

    def print_basic_information(self):
        print(self.__class__.__name__)

class Cat(Animal):

    def __init__(self, animal_type: str, name: str, age: int, weight: int, legs: int, sound:
        super().__init__(animal_type, name, age, weight, legs, sound)

class Snake(Animal):

    def __init__(self, animal_type: str, name: str, age: int, weight: int, legs: int, sound:
        super().__init__(animal_type, name, age, weight, legs, sound)

obj_list = []
with open('./data/animals.csv') as f:
    for line in f.readlines()[1:]:
        animal_type = line.split(", ")[0]
```

```
        name = line.split(", ")[1]
        age = int(line.split(", ")[2])
        weight = int(line.split(", ")[3])
        legs = int(line.split(", ")[4])
        # create objects based on animal_type
        if animal_type == "dog":
            obj = Dog(animal_type=animal_type, name=name, age=age, weight=weight, legs=legs,
        elif animal_type == "cat":
            obj = Cat(animal_type=animal_type, name=name, age=age, weight=weight, legs=legs,
        else:
            obj = Snake(animal_type=animal_type, name=name, age=age, weight=weight, legs=legs
        # add objects to obj_list
        obj_list.append(obj)

for elem in obj_list:
    print("--\n")
    elem.print_basic_information()
    elem.speak()
    c_sum = elem.sum_of_age_and_legs()
    print(c_sum)
```

```
--

Dog
Wuff
14
--

Type = snake and name = masha
Ssss
12
--

Dog
Wuff
12
--

Type = cat and name = leslie
Meww
9
--
```

```
Type = cat and name = dummy
Meww
9
--

Type = cat and name = puffy
Meww
9
--

Type = snake and name = snizzy
Ssss
5
--

Type = snake and name = sazzles
Ssss
2
--

Dog
Wuff
6
--

Type = cat and name = wooly
Meww
9
--

Type = cat and name = beast
Meww
9
```

Exercise Modules

```
'''
Collection of ...FILL HERE...
'''
```

```
module_variable = "Hello"

def reverse(string):
    return string[::-1]

def bi_rev(string):
    return string[::-2]

def say_something():
    return module_variable

def greet_somebody(name):
    return f"Hello {name}"
```

```
#%% Import entire module: string_operations.py
import string_operations as SO

print(SO.__doc__)
#OUT: Collection of methods to manipulate strings.

SO.say_something()
#OUT: Hello

print(SO.module_variable)
#OUT: Hello

my_string = "Hello Universe!"
rev = SO.reverse(my_string)
print(rev)
#OUT: !esrevinU olleH

bi_rev = SO.bi_rev(my_string)
print(bi_rev)
#OUT: !seiUolH

#%% Import specific functions only
from string_operations import reverse
print(reverse("!esrevinU olleH"))
#OUT: Hello Universe!

# greet somebody in reversed order
from string_operations import reverse, greet_somebody
```

```
name = 'Andreas'
print(greet_somebody(reverse(name)))
#OUT: Hello saerdnA
```

Exercise: Jupyter Notebook

> **Note**
>
> We skip the part about creating a Jupyter notebook and go straight to the coding part.

2. Import

```
pip install numpy pandas matplotlib
```

```
# Use of convention aliases
import numpy as np
import pandas as pd

# df means Data Frame
df = pd.read_csv('/home/nils/dev/mscids-notes/hs25/pds/data/airquality.csv')

# Preview
df.head(n=10)
```

	rownames	Ozone	Solar.R	Wind	Temp	Month	Day
0	1.0	41.0	190.0	7.4	67.0	5.0	1.0
1	2.0	36.0	118.0	8.0	72.0	5.0	2.0
2	3.0	12.0	149.0	12.6	74.0	5.0	3.0
3	4.0	18.0	313.0	11.5	62.0	5.0	4.0
4	5.0	NaN	NaN	14.3	56.0	5.0	5.0
5	6.0	28.0	NaN	14.9	66.0	5.0	6.0
6	7.0	23.0	299.0	8.6	65.0	5.0	7.0
7	8.0	19.0	99.0	13.8	59.0	5.0	8.0
8	9.0	8.0	19.0	20.1	61.0	5.0	9.0
9	10.0	NaN	194.0	8.6	69.0	5.0	10.0

3. Stats

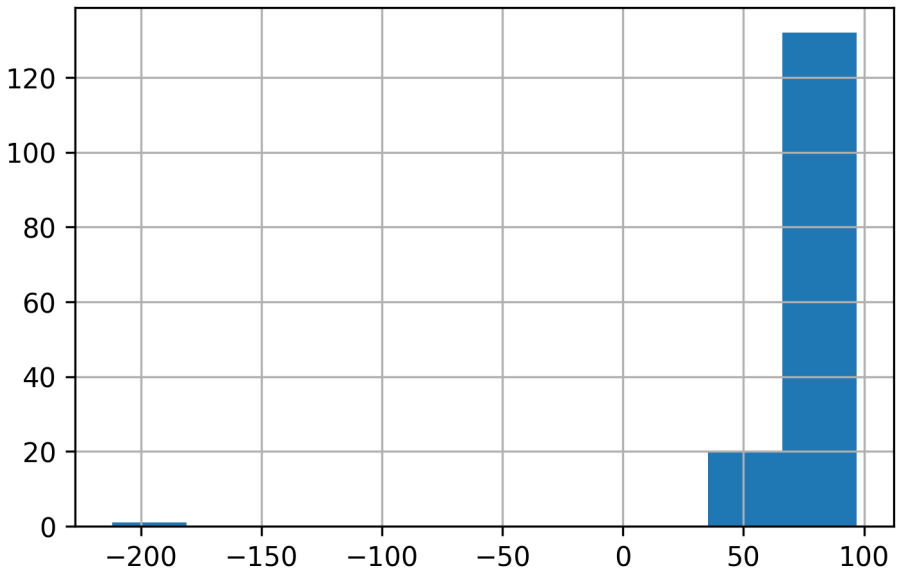
```
# Select column "Temp" from df
df['Temp'].describe()
```

```
count    153.000000
mean      76.124183
std       25.227992
min     -212.000000
25%       72.000000
50%       79.000000
75%       85.000000
max       97.000000
Name: Temp, dtype: float64
```

4. Outlier

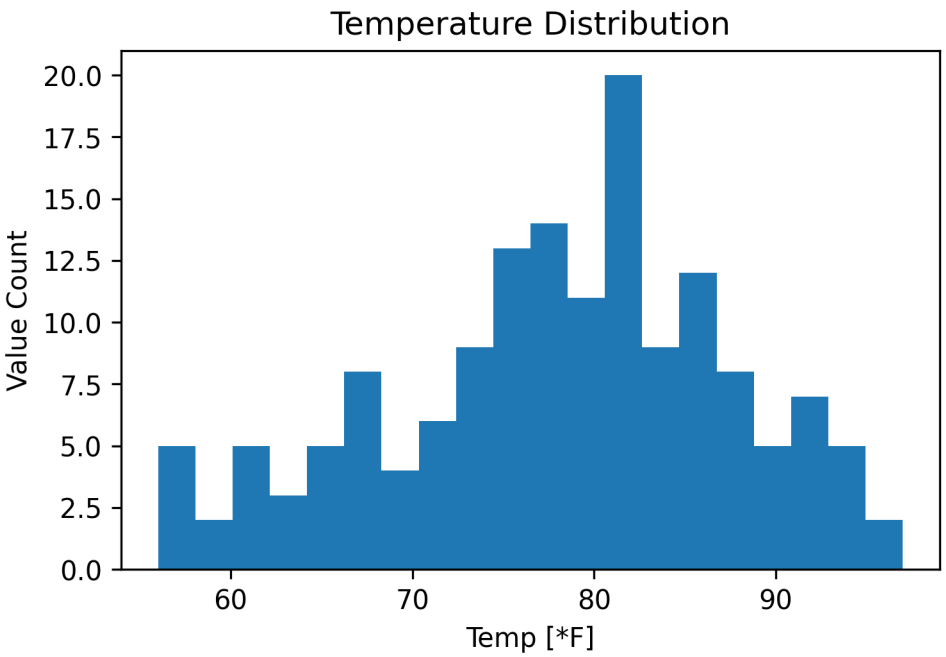
```
# Get overview
df['Temp'].hist() # -200 is a outlier

valid_mean = df[df['Temp'] >= 0]['Temp'].mean() # Mean without outlier
old_temp_mean = np.mean(df['Temp'])

# Correction
df['Temp'] = np.where( (df['Temp'] < 0 ), valid_mean, df['Temp'])

new_temp_mean = np.mean(df['Temp'])

print(f'Old mean {old_temp_mean} vs. new mean {new_temp_mean}')
```

```
Old mean 76.12418300653594 vs. new mean 78.01973684210526
```



5. Plotting

```
from matplotlib import pyplot as plt

fig, ax = plt.subplots() # Main construction

ax.hist(df['Temp'], bins=20) # Main Plot

ax.set_title("Temperature Distribution")
ax.set_xlabel("Temp [*F]")
ax.set_ylabel("Value Count")

plt.show()
```



Note: Since we corrected the outlier, the plot looks pretty good.

6. Define Temp Converter

```
def fahrenheit_to_celsius(temp: int) -> int:
    """ Converts Fahrenheit to Celsius """
    result = np.round((temp-32)*5/9, 1)
    return result
```

7. Fahrenheit to Cerlsius

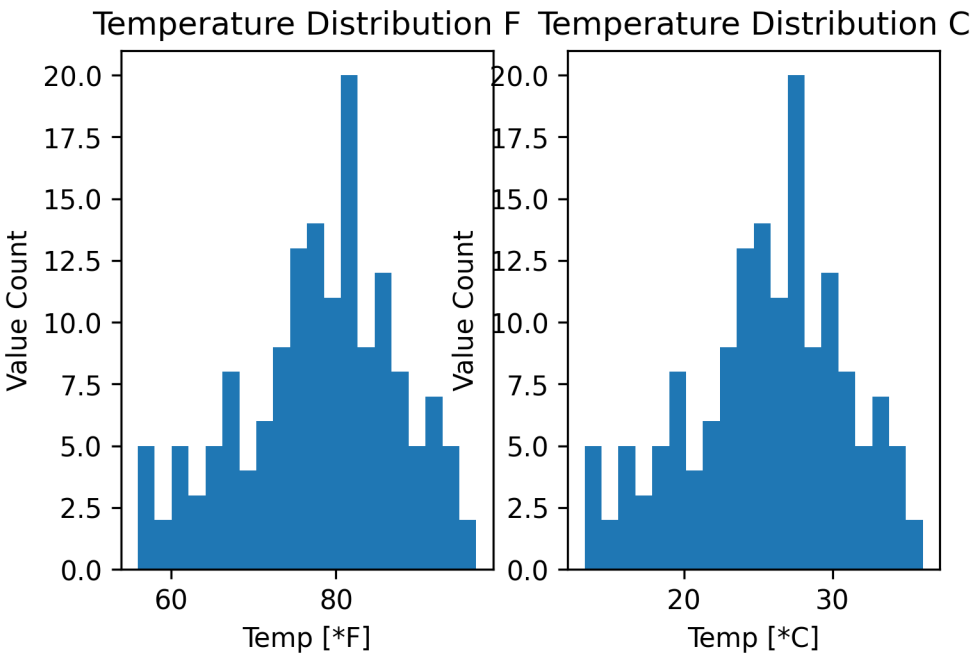
```
# We can directly define a new column
df['Temp_C'] = df['Temp'].map(fahrenheit_to_celsius)
df.head(n=10)
```

	rownames	Ozone	Solar.R	Wind	Temp	Month	Day	Temp_C
0	1.0	41.0	190.0	7.4	67.0	5.0	1.0	19.4
1	2.0	36.0	118.0	8.0	72.0	5.0	2.0	22.2

	rownames	Ozone	Solar.R	Wind	Temp	Month	Day	Temp_C
2	3.0	12.0	149.0	12.6	74.0	5.0	3.0	23.3
3	4.0	18.0	313.0	11.5	62.0	5.0	4.0	16.7
4	5.0	NaN	NaN	14.3	56.0	5.0	5.0	13.3
5	6.0	28.0	NaN	14.9	66.0	5.0	6.0	18.9
6	7.0	23.0	299.0	8.6	65.0	5.0	7.0	18.3
7	8.0	19.0	99.0	13.8	59.0	5.0	8.0	15.0
8	9.0	8.0	19.0	20.1	61.0	5.0	9.0	16.1
9	10.0	NaN	194.0	8.6	69.0	5.0	10.0	20.6

8. Double-Plot

```
fig, ax = plt.subplots(ncols=2) # Define two cols

# First plot (ax[0])
ax[0].hist(df['Temp'], bins=20)
ax[0].set_title("Temperature Distribution F")
ax[0].set_xlabel("Temp [*F]")
ax[0].set_ylabel("Value Count")

# Second plot (ax[1])
ax[1].hist(df['Temp_C'], bins=20)
ax[1].set_title("Temperature Distribution C")
ax[1].set_xlabel("Temp [*C]")
ax[1].set_ylabel("Value Count")

plt.show()
```



9. & 10. Class "AirQualityReport"

```python
class AirQualityReport:
    def __init__(self, df: pd.DataFrame):
        for col_name, data in df.items():
            if pd.api.types.is_numeric_dtype(data):
                mean_value = data.mean()
                setattr(self, f"_{col_name}", mean_value)

    def calculate_max_value(self):
        max_so_far = -float('inf')
        max_column_name = None

        for attr_name, attr_value in self.__dict__.items():
            if attr_name.startswith('_') and isinstance(attr_value, (int, float)):
                if attr_value > max_so_far:
                    max_so_far = attr_value
                    max_column_name = attr_name[1:]

        return {
            "column_name": max_column_name,
```

```
            "max_value": max_so_far
        }

report = AirQualityReport(df)
print(report.__dict__)
```

```
{'_rownames': np.float64(77.0), '_Ozone': np.float64(42.12931034482759), '_Solar.R': np.float
```

Exercise: Comprehension & Lambda I

Task 1.

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

# Task 1: copy all even elements from 'nums' into a new list.
print("\n---- Task 1 ----")
my_list = []
for n in nums:
    if not n%2:
        my_list.append(n)
print(f"for loop: {my_list=}")

# solution with list comprehension:
my_list = [num for num in nums if num % 2 == 0]
print(f"comprehension: {my_list=}")
```

```
---- Task 1 ----
for loop: my_list=[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
comprehension: my_list=[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Task 2.

```
# Task 2: calculate the square of each element in the list 'nums'.
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

print("\n---- Task 2 ----")
my_list = []
```

```
for n in nums:
  my_list.append(n*n)
print(f"for loop: {my_list=}")

# solution with list comprehension:
my_list = [num*num for num in nums]
print(f"comprehension: {my_list=}")
```

```
---- Task 2 ----
for loop: my_list=[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 3
comprehension: my_list=[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 2
```

Task 3.

```
# Task 3: calculate the square of each element in the list 'nums' by applying
# a lambda function on the list (using map function).
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

print("\n---- Task 3 ----")
my_list = []
for n in nums:
  my_list.append(n*n)
print(f"for loop: {my_list=}")

# solution with list comprehension:
my_list = list(map(lambda x: x*x, nums))
print(f"lambda: {my_list=}")
```

```
---- Task 3 ----
for loop: my_list=[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 3
lambda: my_list=[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324
```

Task 4.

```
# Task 4: create nested lists comprising all lower prime numbers up to the
# given number in 'nums'.
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

```
primes = [1, 2, 3, 5, 7, 11, 13, 17, 19, 23]

print("\n---- Task 4 ----")
my_list = []
for n in nums:
  p_list = []
  for p in primes:
    if p <= n:
      p_list.append(p)
  my_list.append(p_list)
print(f"for loop: {my_list=}")

# solution with list comprehension:
my_list = [[p for p in primes if p <= n] for n in nums]
print(f"comprehension: {my_list=}")
```

```
---- Task 4 ----
for loop: my_list=[[1], [1, 2], [1, 2, 3], [1, 2, 3], [1, 2, 3, 5], [1, 2, 3, 5], [1, 2, 3, 5
comprehension: my_list=[[1], [1, 2], [1, 2, 3], [1, 2, 3], [1, 2, 3, 5], [1, 2, 3, 5], [1, 2
```

Exercise: Comprehension & Lambda II

Task 1.

```
# The variable 'chars' provides a list of all possible lower case characters.
# And the method count of a str object counts the number of occurrence for a
# particular object.

import string
chars = string.ascii_lowercase

lorem = """Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed
diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
rebum. Stet clita gubergren, no sea takimata sanctus est Lorem ipsum dolor
sit amet."""

# Task 1: use the map function to apply a lambda function for counting all
# characters in the text.
```

```
count1 = {}
count1 = dict(map(lambda char: (char, lorem.count(char)), chars))

print(count1)
print(count1["e"])
```

```
{'a': 21, 'b': 3, 'c': 6, 'd': 12, 'e': 28, 'f': 0, 'g': 4, 'h': 0, 'i': 15, 'j': 1, 'k': 1,
28
```

Task 2.

```
# Task 2: solve the same problem by means of a dictionary comprehension.
count2 = {}
count2 = {char: lorem.count(char) for char in chars}

print(count2)
print(count2["e"])
```

```
{'a': 21, 'b': 3, 'c': 6, 'd': 12, 'e': 28, 'f': 0, 'g': 4, 'h': 0, 'i': 15, 'j': 1, 'k': 1,
28
```