Introduction to programming

Labor10

Exams

- **Endterm exam** – Next week!!!

- **Retake Midterm – Registration -> Next week**
  - 5th of December 2023, 10:00-16:00
  - IK-205, IK-206

- **Retake Endterm – Registration -> Next week**
  - 7th of December 2023, 10:00-16:00
  - IK-206, IK-207

Revision

- Command line arguments
  - `sys.argv`
- Handling files
  - `open()`
  - `read()`
  - `readline()`
  - `readlines()`
  - `write()`
  - `writelines()`

Homework 1

Number of local maximums

- https://viskillz.inf.unideb.hu/prog/#/?week=P1081&exercise=P108109c&page=sheet

Homework 1

The function

- Write a function named `count_of_local_maximums` that returns the number of items in the list given as a parameter that are greater than both of its neighbours.

Returned value

- number of elements matching the condition

Command Line Arguments:

- `12 34 12 56 5`

Output:

- `2`

Solution – 1

```
import sys

def count_of_local_maximums() -> int:
    numbers = [int(s) for s in sys.argv[1:]]
    count = 0
    for i in range(len(numbers) - 2):
        if numbers[i] < numbers[i + 1] > numbers[i + 2]:
            count += 1
    return count

def main():
    print(count_of_local_maximums())

if __name__ == "__main__":
    main()
```

Solution – 2

```
import sys
def count_of_local_maximums(numbers: list[int]) -> int:
    count = 0
    for i in range(len(numbers) - 2):
        if numbers[i] < numbers[i + 1] > numbers[i + 2]:
            count += 1
    return count
def main():
    numbers = []
    argc = len(sys.argv)
    for i in range(1, argc):
        numbers.append(int(sys.argv[i]))
    print(count_of_local_maximums(numbers))
if __name__ == "__main__":
    main()
```

Homework 2 – Formula 1

- Write a program that takes the name of a text file as a command-line argument.
- The text file contains data about Formula 1 drivers in each line in the following format:

- `driver_name;race_location;lap_completed;placing`

- The program should aggregate the total number of laps completed by each driver. Subsequently, the program should write to the standard output the names of the drivers in descending order based on the total number of laps completed.
- In case two drivers have completed the same number of laps, the output order should be based on the alphabetical order of their names.

Homework 2

- Command Line Argument:
```
sample.txt
```

- Contents of the sample.txt file:
```
Lewis Hamilton;Melbourne;58;1
Daniel Ricciardo;Melbourne;57;6
Lewis Hamilton;Sepang;56;2
Fernando Alonso;Sepang;20;0
```

- The result for the sample:
```
Lewis Hamilton
Daniel Ricciardo
Fernando Alonso
```

Solution

```
import sys
drivers={}

with open(sys.argv[1]) as file:
    for line in file:
        data = line.strip("\n").split(";")
        # print(data)
        if data[0] in drivers:
            drivers[data[0]] += int(data[2])
        else:
            drivers[data[0]] = int(data[2])
    # print(drivers)
for key, value in sorted(drivers.items(),
                         key = lambda x: (-x[1], x[0])):
    print(key)
```

Homework 3

Prime selection

- https://progcont.hu/progcont/100317/?pid=201426

Homework 3 – Prime selection

- Write a program that takes the name of a text file as its first command line argument.
- Each line of the text file contains a sequence of integers, separated by exactly one space, and each line must contain at least one value.
- The program should extract and print to the standard output, for each line, the sequence of prime numbers in ascending order.
- If no prime numbers are found in the input line, the string "NOTHING" should be included in the output.
- See the example output for the exact format!

Homework 3 – Prime selection

- Command Line Argument:

```
sample.txt
```

- Contents of the sample.txt file:

```
1 2 3 4 5 6 7 8 9
25 13 12 10 21 53
3 5 7 3 5 7 3 5 7
2 4 8 16 32
```

- The result of the program (standard output):

```
2, 3, 5, 7
13, 53
3, 5, 7
2
```

Solution – 1

```python
import sys
import math

def is_prime(n: int) -> bool:
    if n == 2:
        return True
    if n < 2 or n % 2 == 0:
        return False
    for i in range(3, int(math.sqrt(n))+1, 2):
        if n % i == 0:
            return False
    return True

def main():
    with open(sys.argv[1]) as file:
        for line in file:
            numbers = line.strip().split(' ')
            primes=[]
            for number in numbers:
                if is_prime(int(number)) and int(number) not in primes:
                    primes.append(int(number))
            primes.sort()
            if primes:
                for i in range(len(primes)):
                    if i < len(primes)-1:
                        print(primes[i], end=', ')
                    else:
                        print(primes[i])
            else:
                print("NOTHING")
if __name__ == '__main__':
    main()
```

Solution – 2

```
import sys
import math

def is_prime(n: int) -> bool:
    if n == 2:
        return True
    if n < 2 or n % 2 == 0:
        return False
    for i in range(3, int(math.sqrt(n))+1, 2):
        if n % i == 0:
            return False
    return True
def main():
    with open(sys.argv[1]) as file:
        for line in file:
            numbers = line.strip().split(' ')
            primes = []
            for number in numbers:
                if is_prime(int(number)) and int(number) not in primes:
                    primes.append(int(number))
            primes.sort()
            if primes:
                print(', '.join(str(p) for p in primes))
            else:
                print("NOTHING")
if __name__ == '__main__':
    main()
```

Tuple

- Ordered storage
- Index to refer to individual items
- One element/value can occur more than once
- Can store multiple types at the same time
- The elements are given in round brackets
- Cannot be changed (**immutable**):
  - Elements
  - Order of elements
  - Number of elements

Tuple

- An immutable object storing a fixed number of data items.
- `t = (5, 9)`
- `print(t[0], t[1])   # 5  9`
- `type(t)`
- `<class 'tuple'>`

- The tuple type object is created with the comma-separated values listed in round brackets.
- And each data item is accessed by indexing, t[0] is the first data item (here 5) and t[1] is the second (here 9).
- Objects of the tuple type are immutable; the references they store cannot change, neither their number nor their value.

Tuple – Example

```
a = ()
#An empty tuple a is created using parentheses ().
print(a)

b = tuple()
#An empty tuple b is created using the tuple() constructor.
print(b)

c = 1,
print(c)
#One element tuple
```

Output

```
()
()
(1,)
```

Tuple – Example

- Calculate the sum of the tuple elements:

```
def sum(test_tuple):
    # Converting into list
    test = list(test_tuple)
    count = 0
    for i in test:
        count += i
    return count

test_tuple = (5, 20, 3, 7, 6, 8)
print(sum(test_tuple))
```

Tuple – Example

- Calculate the sum of the tuple elements:

```
def sum2(test_tuple):
# Convert the tuple to a list using a list
comprehension
    test = [x for x in test_tuple]
    return sum(test)

test_tuple = (5, 20, 3, 7, 6, 8)
print(sum2(test_tuple))
```

Tuple – Example

- Modify the tuple:

```
t1 = (10, 20, 30, 40, 50)
print("Original Tuple :", t1)
l = list(t1) # Converting into list
l[2] = 33
t1 = tuple(l) #Converting into tuple
print("Modified Tuple :", t1)
```

Tuple

- Often, several pieces of data are placed next to each other in one program site, often only temporarily.
- For example, you might want to make a function with two return values that finds the minimum and maximum of a list at the same time.
- Or we would work with pairs of numbers that represent ranges of lists (start index, end index).
- Perhaps we would store the coordinates of a point (x, y), but we don't need string conversion, operators, anything else that would make it important to define a separate class.
- In this case, we need a container.

Tuple

- But it's not a list, because that means something a bit different: the number of items in the list can change and they can be swapped.

- This is not true in a function with two return values: there the first data returned is the minimum of the list, the second the maximum – they are not mutable (they have different meanings) and cannot change in number (there will always be two).

Tuple – Example

- Write the function that returns the minimum and maximum elements of a list.

```
def minmax(numbers: list[int]) -> tuple:
    min = max = numbers[0]
    for i in range(1, len(numbers)):
        if numbers[i] < min: min = numbers[i]
        if numbers[i] > max: max = numbers[i]
    return (min, max)
```

Tuple – Example

```python
competition = [
    (4, "Peter"),
    (3, "Mike"),
    (2, "David"),
    (1, "George"),
    (4, "Rudolf"),
]

for place, name in sorted(competition):
    print(f"{place}. place: {name}")
```

Tuple

- Output:

```
1. place: George
2. place: David
3. place: Mike
4. place: Peter
4. place: Rudolf
```

- The loop iterates through this container, unpacking each tuple into variables named name and place.
- In addition, the tuples are sorted first: this is provided by the sorted() function.
- Apparently, we are not actually iterating over the original container, but over a sorted version of it.
- In sorting, the sorted() function had to compare tuple elements. These are interpreted by the comparison relational operators. The comparison of tuples proceeds one by one according to the data element.

Named tuple

- A **named tuple** is an extension and custom data type that enrich built-in tuples with extra utilities.
- They are very useful in context where we need to create a data structure that can be accessed by both the positional index and the named attribute of the elements.

```
from typing import NamedTuple

# Declaring namedtuple()
Student = NamedTuple('Student', [('name', str), ('age', int),
('neptun', str)])
# Adding values
S = Student('Mike', 19, 'NDKYCF')
# Access using index
print("Student age: ",S[1])
# Access using name
print("Student name: ", S.name)
# Access neptun code
print("Neptun code: ", S.neptun)
```

Minions

https://viskillz.inf.unideb.hu/prog/#/?week=P1083&exercise=P108301e&page=sheet

- The history of the minions date back to the beginning of time.
- Minions started as yellow single-celled organisms and evolved through the ages, always serving the most despicable masters.
- Since they constantly lost these masters — from T-Rex to Napoleon — the minions now have no one to serve, and they have fallen into deep depression.
- In this task, you need to read the data of the minions from the standard input, sort them, and then write them to the standard output.

Minions – Formats

- On the standard input, a minion appears in the following format:

```
    minion_name;hunger;motivation;pants_size
```

- On the standard output, a minion appears in the following format:

```
    minion_name hunger (pants_size)
```

Notations:

- `minion_name`: a unique string up to 30 characters long containing only English letters
- `hunger`: a non-negative integer value
- `motivation`: a non-negative integer value
- `pants_size`: one of the strings 'S', 'L', 'XL' and 'XXL

Minion – Named Tuple

- Define a **Minion** Named Tuple with the following fields:
  - **`name`**: name of the minion **`(type: <class 'str'>)`**
  - **`hunger`**: hunger of the minion **`(type: <class 'int'>)`**
  - **`motivation`**: motivation of the minion **`(type: <class 'int'>)`**
  - **`size:`** size of the minion **`(type: <class 'str'>)`**

Named Minion Tuple

```
from typing import NamedTuple

Minion=NamedTuple("Minion",[("name", str),
                            ("hunger", int),
                            ("motivation", int),
                            ("size", str)])
```

The line_to_minion() function

- Write a function named `line_to_minion()` that takes a line of input and returns a tuple named Minion filled with data extracted from the line.

- **Parameter list**
  - line of input describing a Minion record

- **Returned value**
  - the Minion record corresponding to the line parameter

The line_to_minion() function

```
def line_to_minion(line):
    data = line.strip().split(";")
    return Minion(data[0], int(data[1]),
                  int(data[2]), data[3])
```

The minion_to_line() function

- Write a function called `minion_to_line()`, which gets a Minion record and returns the corresponding string to write to the output.

- **Parameter list**
  - minion – record of type Minion

- **Returned value**
  - string representation corresponding to the minion parameter

The minion_to_line() function

```
def minion_to_line(minion):
    return f"{minion.name} {minion.hunger}
             ({minion.size})"
```

The sort_minions() function

- Write a function named `sort_minions()` that gets a list of Minions, then sorts and returns it.

Parameter list

- minions – list of Minion items

Returned value

- the list of minions received as a parameter, sorted according to the following criteria:
  - in descending order of motivation
  - in increasing order by name

The sort_minions() function

```
def sort_minions(minions):
    minions.sort(key = lambda minion:
    (-minion.motivation, minion.name))

    return minions
```

The main() function

- Create a main program that makes the solution testable based on the following specifications!

- The standard input lines contain the data of a Minion separated by semicolons (;).

- The end of the test cases is indicated by an end-of-file (EOF).

- The read lines are processed as follows:
  - the Minion list is generated using the `line_to_minion()` function
  - sort the Minion list using `sort_minions()` function
  - print the Minion list using the `minion_to_line()` function

The main() function

```
def main():
     minions = []

     for line in sys.stdin:
          minions.append(line_to_minion(line))

     for minion in sort_minions(minions):
          print(minion_to_line(minion))
```

Input/Output

Sample Input:

- `Bob;87;100;S`
- `Dave;43;10;L`
- `Stuart;50;30;L`
- `Jerry;40;20;XL`

Output for Sample Input:

- `Bob 87 (S)`
- `Stuart 50 (L)`
- `Jerry 40 (XL)`
- `Dave 43 (L)`

```python
import sys
from typing import NamedTuple

Minion = NamedTuple("Minion", [("name", str), ("hunger", int), ("motivation", int),
("size", str)])

def minion_to_line(minion):
    return f"{minion.name} {minion.hunger} ({minion.size})"

def line_to_minion(line):
    data = line.strip().split(";")
    return Minion(data[0], int(data[1]), int(data[2]), data[3])

def sort_minions(minions):
    minions.sort(key=lambda minions: (-minions.motivation, minions.name))
    return minions

def main():
    minions = []
    for line in sys.stdin:
        minions.append(line_to_minion(line))

    for minion in sort_minions(minions):
        print(minion_to_line(minion))

if __name__ == '__main__':
    main()
```

The main() function – Read from file
https://viskillz.inf.unideb.hu/prog/#/?week=P1094&exercise=P109401e&page=sheet

Sample Input:
sample.txt – Command line argument

```
Bob;87;100;S
Dave;43;10;L
Stuart;50;30;L
Jerry;40;20;XL
```

Output for Sample Input:

```
Bob 87 (S)
Stuart 50 (L)
Jerry 40 (XL)
Dave 43 (L)
```

The main() function

```
def main():
    with open(sys.argv[1]) as file:
        minions=[]
        for line in file:
            minions.append(line_to_minion(line))

        for minion in sort_minions(minions):
            print(minion_to_line(minion))

if __name__ == '__main__':
    main()
```

Exercise – RollerCoaster https://viskillz.inf.unideb.hu/prog/#/?week=P1083&exercise=P108302e&page=sheet

- In a crowded theme (amusement) park, you can queue for hours to get on a roller coaster.
- Given the size of the parks and the queues, it's worth planning the order in which you'll get on the rides.
- PortAventura World is one of Europe's largest theme parks, located on the Costa Brava between Salou and Tarragona.
- Its classic area (apart from the continent's only Ferrari park) contains several themed worlds, each with at least one roller coaster.
- In this exercise, you need to read roller coaster data from the standard input, sort them, and then print the result to the standard output.

Exercise – RollerCoaster

Formats:

A standard input contains a roller coaster in the following format:

`<coaster_name>;<world_name>;<minimum_height>;<wait_time>.`

The standard output is a roller coaster in the following format:

`<coaster_name> (<world_nave>): <wait_time>`

Notations:

- **rollercoaster_name:** a unique string up to 30 characters long, containing only English letters and spaces
- **world_name:** a unique string up to 30 characters long containing only English letters and spaces
- **minimum_height:** a non-negative integer value
- **wait_time:** a non-negative integer value
- The solution to the problem must be broken down into several functions, each of which is described below.

Input/Output

- **Sample Input**

```
Furius baco;Polynesia;140;120
Shambhala;China;140;120
Dragon Khan;China;140;80
Stampida;Far West;120;20
Tami Tami;SesamoAventura;100;20
El Diablo;Mexico;140;30
```

- **Output for Sample Input**

```
Stampida (Far West): 20
Tami Tami (SesamoAventura): 20
El Diablo (Mexico): 30
Dragon Khan (China): 80
Furius baco (Polynesia): 120
Shambhala (China): 120
```

RollerCoaster Named Tuple

- Define a **RollerCoaster** named tuple with the following fields:
  - `name`: name of the roller coaster (`type: <class 'str'>`)
  - `world`: name of the world (`type: <class 'str'>`)
  - `height`: the minimum height (`type: <class 'int'>`)
  - `time`: the waiting time (`type: <class 'int'>`)

RollerCoaster Named Tuple

```
from typing import NamedTuple

RollerCoaster = NamedTuple("RollerCoaster",
[("name", str), ("world", str),
("height", int), ("time", int)])
```

The line_to_coaster() function

- Write a function named **line_to_coaster()** that takes a line of input and returns a tuple named RollerCoaster filled with data extracted from the line.

- Parameter list
  - the input is a line describing a RollerCoaster record
- Returned value
  - the RollerCoaster record corresponding to the line parameter

The line_to_coaster() function

```
def line_to_coaster(line):
    data = line.strip().split(";")
    return RollerCoaster(data[0], data[1],
           int(data[2]), int(data[3]))
```

The coaster_to_line() function

- Write a function called `coaster_to_line()` that gets a RollerCoaster record and returns the corresponding string to write to the output.

- **Parameter list**
  - coaster – record of type RollerCoaster

- **Returned value**
  - string representation corresponding to the coaster parameter

The coaster_to_line() function

```
def coaster_to_line(coaster):

    return f"{coaster.name}
    ({coaster.world}): {coaster.time}"
```

The sort_coasters() function

- Write a function named `sort_coasters()` that gets a list of RollerCoasters, then sorts and returns it.
- **Parameter list**
  - coasters – list of RollerCoaster items
- **Returned value**
  - the list of coasters received as a parameter, sorted by the following criteria:
    - Increasing order of waiting time
    - descending order by minimum height
    - increasing order by coaster name

The sort_coasters() function

```
def sort_coasters(coasters):

     coasters.sort(key=lambda
     coaster: (coaster.time, -coaster.height,
                coaster.name))

    return coasters
```

```python
import sys
from typing import NamedTuple

RollerCoaster = NamedTuple("RollerCoaster", [("name", str), ("world", str),
                          ("height", int), ("time", int)])

def line_to_coaster(line):
    data = line.strip().split(";")
    return RollerCoaster(data[0], data[1], int(data[2]), int(data[3]))

def coaster_to_line(coaster):
    return f"{coaster.name} ({coaster.world}): {coaster.time}"

def sort_coasters(coasters):
    coasters.sort(key = lambda coaster: (coaster.time, -coaster.height,
                  coaster.name))
    return coasters
def main():
    coasters = []
    for line in sys.stdin:
        coasters.append(line_to_coaster(line))

    for coaster in sort_coasters(coasters):
        print(coaster_to_line(coaster))
if __name__ == '__main__':
    main()
```