Introduction to programming

Labor06

Midterm test

- **Next week!!!**
- 16-20th of October 2023
- During the lessons
- Labor01-Labor05
- Mixed:
  - Paper based (around 30 minutes)
  - Computer based- you are not allowed to use your own laptop!!! (around 60 minutes)
- Professional Days (12th-13th of October 2023)
  - 4 lectures = 1 missed lab

Revision

- GCD, LCM, Relative primes
- Random numbers
  - `random()`
  - `randint()`
- Lists
  - Slicing
  - Operators
  - Functions
- List Comprehension

Homework 1 – Neanderthal gene
https://progcont.hu/progcont/100350/?pid=201515

- A recent study suggests that the risk of coronavirus infection is more severe in people who carry a small Neanderthal gene. Reading this makes you stop for a moment and wonder if you have even an ounce of the Neanderthal gene in you.
- If you make a model, Neanderthal genes are relatively easy to identify. Suppose we describe **a DNA molecule** by a sequence of integers. **If we can find at least three consecutive values in this sequence that form a strictly monotonic series, we can say that the DNA molecule contains a Neanderthal gene.**
- Write a program that reads at least one positive integer per line from the standard input to the end of file (EOF). Each line describes a DNA molecule. If the DNA molecule under test contains a Neanderthal gene, your program should write a line containing a "YES" string to the standard output, otherwise a "NO" string.

Homework 1 – Neanderthal gene

https://progcont.hu/progcont/100350/?pid=201515

Sample Input

```
5 6 7
9 8 7 6 5
2 3
7 11 8 10 9
4 4 4 4
9
1 2 3 1 2
7 8 7 8 7 8 9
```

Output for Sample Input

```
YES
YES
NO
NO
NO
NO
YES
YES
```

Solution – Neanderthal gene

```
while True:
    try:
        numbers =  input().split()
        n = len(numbers)
        DNS = False
        for i in range(n - 2):
            if int(numbers[i]) < int(numbers[i + 1]) < int(numbers[i + 2]) or
               int(numbers[i]) > int(numbers[i + 1]) > int(numbers[i + 2]):
                DNS = True
        if DNS:
            print("YES")
        else:
            print("NO")
    except EOFError:
        break
```

Homework 2 – Piggy bank
https://progcont.hu/progcont/100353/?pid=201527

- Children love to collect their savings in piggy banks. In this exercise, you have to make a statement of these savings.
- Write a program that reads the names of the children from standard input to end-of-file (EOF) and the money they have saved from week to week, line by line!
- The form of a line is:
- `name:quantity[ quantity]...`
- Your program will write in the first line of the standard output how much money the children have saved in total according to the statement! While processing the data, also count the number of children who regularly saved at least 20 forints in their piggy bank every time! Print the latter value on the second line of the standard output!

Homework 2 – Piggy bank

https://progcont.hu/progcont/100353/?pid=201527

Sample Input

- Mark:20 30 40 50 60
- Jogh:25 15 25 15 25 15
- Peter:30
- David:20 30 10 30 20 10 20 30
- Julie:19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Output for Sample Input

- 710
- 2

Solution – Piggy bank

```
sum = 0
c = 0
while True:
    try:
        line = input().split(':')
        money = line[1].split(' ')
        twenty = True
        for i in range(len(money)):
            sum += int(money[i])
            if int(money[i]) < 20:
                twenty = False
        if twenty == True:
            c += 1

    except EOFError:
        break
print(sum)
print(c)
```

Homework 3 – Reverse numbers

https://progcont.hu/progcont/100348/?pid=201508

- Write a program that reads a sequence of numbers from the standard input and writes its elements in reverse order to the standard output.
- Each line of the input contains at least one integer, and the consecutive numbers are separated by exactly one space.
  The first number (n) in each line is the number of elements in the number line to be processed.
  The program must write the following sequence of n integers in reverse order to the standard output. The end of the input is indicated by a line containing only one 0.
  The program need not process this line.
- Make sure that the numbers written in a line are separated by exactly one space, and do not print spaces at the beginning and end of the line.

Homework 3 – Reverse numbers

https://progcont.hu/progcont/100348/?pid=201508

Sample Input

- 4 1 2 3 4
- 5 2 5 3 4 1
- 2 2 3
- 0

Output for Sample Input

- 4 3 2 1
- 1 4 3 5 2
- 3 2

Solution 1 – Reverse numbers

```
line = input().split()

while True:
    n = int(line[0])
    if n == 0:
        break
    i = n
    while i > 1:
        print(int(line[i]), end=' ')
        i -= 1
    print(int(line[i]))

    line = input().split()
```

Solution 2 – Reverse numbers

```
line = input().split()
while line[0] != '0':
    rev = line[:0:-1]
    revstr = ' '.join(x for x in rev)
    print(revstr)
    line = input().split()
```

Solution 3 – Reverse numbers

```
line = input().split()
while line[0] != '0':
    reversenum = list(reversed(line[1:]))
    print(*reversenum) #unpacking the list
    line = input().split()
```

Subroutines

A separate unit of the algorithm.

A tool for reuse and abstraction:

- **Reuse**: it is not necessary to write the same sequence of instructions in several places in the code. We can train the activity to work as a simple instruction.
  An example is embedding.
- **Abstraction**: not just one activity, but a set of similar activities can be used, with parameters to specify the details.
  A representation of generalization.

Subroutines

Two types of subroutines:

- **Function**: set of instructions to produce a value (the return value) somewhere in the code.
  `return` instruction is included in the code, which returns the value given afterwards.
  E.g. What is the cosine of 45°?
- `x = math.cos(45)`

- **Procedure**: set of actions to do something at a given point in the code (**no return value!!!**)
  E.g. Write your name!
  `print(”David”)`

Subroutines – Python

- Functions
  - Built-in functions: `len()` – length of string
  - They are grouped into modules – E.g. `sqrt()`
    - `import math`
    - `math.sqrt(x)` – square root of x
  - Define your own function
- Procedures
  - (They have no return value)
  - Also called a function in Python

Functions – Python

```
def function_name(paramters):
    statements
```

- The **function name** can be arbitrary, the same rule applies as for variable names (it cannot be a Python keyword).
- The statements that form the function body must be indented.

Functions – Python

- The **header** (the first line) starts with the **def** keyword, followed by the function name, followed by the formal parameter list between round brackets, and finally a colon.
- The **formal parameter** list can be empty, or multiple parameters separated by commas.
- The round brackets must always be filled in.
- The **function body** contains instructions that perform the function's task.

Functions without parameters

Example

```
def table7() -> None:
    n = 1
    while n < 11:
        print(n * 7, end=" ")
        n = n + 1

table7()
```

Functions without parameters

Example:

```
def table(base: int) -> None:
    n = 1
    while n < 11:
        print(n * base, end=" ")
        n = n + 1

table(13)
```

Multiple parameter function

Example:

```
def table(base: int, start: int, end: int) -> None:
    print("Part of the multiplication table of {}:".format(base))
    n = start
    while n <= end:
        print("{0} x {1} = {2}".format(n, base, n * base))
        n = n + 1

table(2, 1, 10)
table(8, 10, 20)
```

Local and global variables

- If you define variables in a function body, only the function itself has access to those variables.
- We say that these variables are **local variables** for the function.
- This is the case, for example, for the variables base, start, end and n in the previous exercise.
- Variables defined outside a function are **global variables.**
- Their contents are visible from inside the function, but the function cannot modify them.

Global variables

- Within a function, the **global** statement allows you to indicate inside a function definition which variables should be treated as global variables.

```
def increase()-> None:
    global a
    a = a + 1
    print(a)

a = 15
increase()
```

„Real" Functions

- The **return** statement defines what the value returned by the function should be.
  In this case, it is the set of arguments passed when the function is called.
  Example:

```
def cube(a: int) -> int:
    return a * a * a

b = cube(9)
print(b)
```

Optional parameters

- When defining Python functions, you can specify so-called **optional parameters.**
- This is used when there is a parameter to the function that is important to be a parameter because it is changed from time to time, but in the vast majority of cases it will have the same value.
- In this case, we can give the parameter a default value, and if the function does not receive that parameter when the function is called, it will use the default value.

Optional parameters

- Example

```
def quadrilateral(a: int, b: int = 0) -> int:
    if b == 0:
        b = a
    return a * b

print('Area of square: ', quadrilateral(10))
print('Area of rectangle: ', quadrilateral(10, 12))
```

Optional parameters

- Parameter passing by position

```
print('Area of square: ', quadrilateral(10))
print('Area of rectangle: ', quadrilateral(10, 12))
```

- Parameter passing by name

```
print('Area of square: ', quadrilateral(a=10))
print('Area of rectangle: ', quadrilateral(b=2, a=12))
#the order of the parameters is reversible
```

Optional parameters

- Logically, only optional parameters can be omitted, all normal parameters must be given a value, either by name or by position.

- If you want to specify parameters by name and by position, they must always come first by position.

Exercise

- Write a function that greets the user in the specified language, Hungarian by default, but changes the language if a country code is specified.
- If the country code is not known, let the user know.

Solution

```
def great(name: str, country_code: str = "HU"):
    match country_code:
        case "HU":
            print("Szia {}!".format(name))
        case "EN":
            print("Hi {}!".format(name))
        case "FI":
            print("Moi {}!".format(name))
        case _:
            print("Country code not found!")
```

Parameter passing types

- **By value** (full isolation): A copy of the original variable is made. Functional operations only affect the copy, not the original variable.
- **Reference** (full linkage) The original variable is essentially the same as the variable in the function. Any change affects the original.
- **In Python**: object-reference based
- The original variable and the variable in the function are not the same, but they point to the same object.
- I can change the object in the function (provided it is not immutable), but I cannot change the original variable to point to a different object.

Example

```
def modify(func):
    func[0] = 2  # modify object
    func = [3]  # modify variable


original = [1]
modify(original)
print(original)  # 1=Value 2=Object-reference 3=Reference
```

Main function

- Entry point to the programme:

```
def main():
    pass

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

Exercise

- Write functions that:
  - Fills an empty list with 10 random integers between [1,100]
  - Returns the minimum element in the list
  - Returns the index of the maximum element in the list
  - Returns the number of even elements in the list
  - Returns whether an integer is prime or not (True/False)
  - Returns the number of primes in the list

```python
import random
import math

def generate(li: list) -> None:
    for i in range(10):
        li.append(random.randint(1, 100))

def minimum_value(li: list) -> int:
    min = li[0]
    for i in range(len(li)):
        if li[i] < min:
            min = li[i]
    return min

def maximum_index(li: list) -> int:
    maxi = 0
    for i in range(len(li)):
        if li[i] > li[maxi]:
            maxi = i
    return maxi

def evens(li: list) -> int:
    even = 0
    for i in range(len(li)):
        if li[i] % 2 == 0:
            even += 1
    return even
```

```python
def isprim(n: int) -> bool:
    if n == 0 or n == 1:
        return False
    d = 2
    while d <= math.sqrt(n):
        if n % d == 0:
            return False
        d += 1
    return True

def prime_numbers(li: list) -> int:
    prime = 0
    for i in range(len(li)):
        if isprim(li[i]):
            prime += 1
    return prime

def main():
    li = []
    generate(li)
    print(li)
    print("Minimum value = ", minimum_value(li))
    print("Maximum index = ", maximum_index(li))
    print("Number of evens = ", evens(li))
    print("Number of primes = ", prime_numbers(li))

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