Introduction to programming

Labor08

Revision

- Recursive functions

Homework 1

https://viskillz.inf.unideb.hu/prog/#/2022?week=P1041

- Write a function named *count_of_squares()* that returns the number of squares of the elements in the list given as a parameter.

- In case of the *main()* function, read lists elements until the given condition, and call the *count_of_squares()* function and print out the number of squares elements.
  - Reads lists until EOF
  - Reads n test cases (lists)
  - Reads until an empty line

Count of squares

```
def count_of_squares(numbers: list[int]) -> int:
    count = 0
    for number in numbers:
        if number == int(math.sqrt(number)) ** 2:
            count += 1

    return count
```

Count of squares – Until EOF

```python
import sys
import math

def count_of_squares(numbers: list[int]) -> int:
    count = 0
    for number in numbers:
        if number == math.sqrt(number) ** 2:
            count += 1
    return count

def main():
    for line in sys.stdin:
        numbers = line.split(" ")
        lists = []
        for number in numbers:
            lists.append(int(number))
        print(count_of_squares(lists))
if __name__ == '__main__':
    main()
```

Count of squares – N test cases

```python
import math

def count_of_squares(numbers: list[int]) -> int:
    count = 0
    for number in numbers:
        if number == math.sqrt(number) ** 2:
            count += 1
    return count

def main():
    n = int(input())
    for i in range(n):
        numbers = input().split(" ")
        lists = []
        for number in numbers:
            lists.append(int(number))
        print(count_of_squares(lists))
if __name__ == '__main__':
    main()
```

Count of squares – Until an empty line

```python
import sys
import math

def count_of_squares(numbers: list[int]) -> int:
    count = 0
    for number in numbers:
        if number == math.sqrt(number) ** 2:
            count += 1
    return count

def main():
    for line in sys.stdin:
        if line.strip() == "":
            break
        numbers = line.split(" ")
        lists = []
        for number in numbers:
            lists.append(int(number))
        print(count_of_squares(lists))

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

Homework 2

https://viskillz.inf.unideb.hu/prog/#/2022?week=P1051

- Write a function named *delete_even_digits()* that returns the input string without even digit characters.

- In case of the *main()* function, read strings until the given condition, and call the *delete_even_digits()* function and print out the new string without even digits.
  - Reads strings until EOF
  - Reads n test cases
  - Reads until an empty string

Delete even digits

```python
def delete_even_digits(original: str) -> str:
    return "".join([c for c in original
            if c not in "02468"])
```

Delete even digits – Until EOF

```
import sys

def delete_even_digits(original: str) -> str:
    return "".join([c for c in original
           if c not in "02468"])

def main():
    for line in sys.stdin:
        print(delete_even_digits(line.strip()))

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

Delete even digits – N test cases

```python
def delete_even_digits(original: str) -> str:
    return "".join([c for c in original
           if c not in "02468"])

def main():

    n = int(input())
    for i in range(n):
        line = input()
        print(delete_even_digits(line))

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

Delete even digits – Until an empty line

```python
import sys

def delete_even_digits(original: str) -> str:
    return "".join([c for c in original
           if c not in "02468"])

def main():
    for line in sys.stdin:
        if line.strip() == "":
            break
        print(delete_even_digits(line.strip()))
```

Homework3

https://viskillz.inf.unideb.hu/prog/#/?week=P1032

- Write a *Pythagorean()* function that returns the hypotenuse of a triangle given the two side (a and b).

- In case of the *main()* function, read datas until the given condition (a b ->in one line, seperated by the space character), and call the *Pythagorean()* function and print out the hypotenuse value.
  - Reads datas until EOF
  - Reads n test cases
  - Reads until an empty line

Pythagorean() function

```
import math


def Pythagorean(a: int, b: int) -> float:
    return math.sqrt(a**2 + b**2)
```

Pythagorean() function – Until EOF

```
import math
import sys

def Pythagorean(a: int, b: int) -> float:
    return math.sqrt(a**2 + b**2)

def main():
    for line in sys.stdin:
        numbers=line.split()
        a=int(numbers[0])
        b=int(numbers[1])
        print('{0:.2f}'.format(Pythagorean(a,b)))

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

Pythagorean() function – Until EOF

```python
import math
import sys

def Pythagorean(a: int, b: int) -> float:
    return math.sqrt(a**2 + b**2)

def main():
    for line in sys.stdin:
        numbers=line.split()
        a=int(numbers[0])
        b=int(numbers[1])
        print('{0:.2f}'.format(Pythagorean(a,b)))

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

Pythagorean() function – N test cases

```
import math
import sys

def Pythagorean(a: int, b: int) -> float:
    return math.sqrt(a**2 + b**2)

def main():
    n = int(input())
    for i in range(n):
        numbers = input().split()
        a=int(numbers[0])
        b=int(numbers[1])
        print('{0:.2f}'.format(Pythagorean(a,b)))

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

Pythagorean() function – Until an empty line

```python
import math
import sys

def Pythagorean(a: int, b: int) -> float:
    return math.sqrt(a**2 + b**2)

def main():
    for line in sys.stdin:
        if line.strip() == "":
            break
        numbers=line.split()
        a=int(numbers[0])
        b=int(numbers[1])
        print('{0:.2f}'.format(Pythagorean(a,b)))

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

SET

- Sets are used to store multiple items in a single variable.
- The set type: **class set**
- The Python language has a built-in set type, the **class set.**
- It supports the usual set operations:
  - insert an element,
  - remove an element,
  - intersection,
  - union,
  - difference
  - symmetric difference of sets.

SET

- Sets are written with curly brackets.

Example

Create a Set:

-
  ```
  s = {"apple", "banana", "cherry"}
  print(s)
  ```

- Set items/elements are **unordered**, **unchangeable**, and do **not allow duplicate values**.

SET

```
s = set(["hungarian", "english", "german"])  # 3 element
set
s = {"hungarian", "english", "german"}
s = set()  # empty set, attention: {} means -> dictionary!

# queries: in, not in
print("english" in s)
print("japanese" not in s)

# modifiers: add, remove
s.add("italian")
s.remove("german")
```

SET operations

- s1 & s2 intersection
- s1 | s2 union
- s1 – s2 difference
- s1 ^ s2 delta/symmetric difference
- The & AND operator is the intersection: it looks for elements that are included in the first AND the second set.
- The | OR operator is the union, looking for elements that are in the first OR the second set.
- The – difference, takes the elements found in the second set from the first set.
- The delta operation is denoted by the ^ (hat, caret) operator. This returns elements that are in only in the first set or only the second set (like the XOR operation).

Example

```
s1 = {"hungarian", "english", "german"}
s2 = set()
s2.add("italian")
s2.add("german")
print(s1)
print(s2)
print("-----------------------------------------")
# Set operations: intersection, union, difference,
symmentric difference (delta, xor)
print(s1 & s2)
print(s1 | s2)
print(s1 - s2)
print(s1 ^ s2)
```

SET operations

SET operations	SET operations names
|	Union
&	Intersection
–	Difference
^	Symmetric Difference

```python
s1 = {"hungarian", "english", "german"}
s2 = set()
s2.add("italian")
s2.add("german")
print(s1)
print(s2)
print("--------------------------------")
# Set operations: intersection, union, difference,
symmentric difference (delta, xor)
print(s1.intersection(s2))
print(s1 & s2)
print(s1.union(s2))
print(s1 | s2)
print(s1.difference(s2))
print(s1 - s2)
print(s1.symmetric_difference(s2))
print(s1 ^ s2)
```

SET operations

- **len(s):** returns the length of the set s
- **x in s:** returns true if x is a member of set s
- **x not in s:** returns true if x is not an element of set s
- **s1.isdisjoint(s2):** returns true if the intersection of s1 and s2 is an empty set (disjoint sets)
- **s1.issubset(s2):**
  - s1 <= s2: returns true if s1 is a subset of s2
  - s1 < s2: returns true if s1 is a real subset of s2

SET operations

- **s.add(element)**: adds element to set s.
- **s.remove(element)**: removes element from set. KeyError exception thrown if element not found in s.
- **s.discard(element)**: removes element from set if it is in the set.
- **s.pop()**: remove and return an arbitrary element from the set.
- **s.clear()**: clear all elements from the set.
- **s.copy()**: creates a copy of the s set

Exercise

- Fill two sets with random integer numbers between 1 and 10. Read the size of the sets from the standard input.
- Print the union, intersection, difference and symmetric difference of the two sets.

Solution

```python
import random
s1 = set()
s2 = set()
n=int(input("n = "))
m=int(input("m = "))
while len(s1)<n:
    s1.add(random.randint(1,10))
while len(s2)<m:
    s2.add(random.randint(1,10))
print(s1)
print(s2)
print("Union =", s1.union(s2))
print("Intersection =", s1.intersection(s2))
print("Difference =", s1.difference(s2))
print("Symmetric difference=", s1.symmetric_difference(s2))
```

Exercise

- Fill two sets with randomly generated odd and even numbers between 1 and 20, then print the union of the generated sets.
- Read the sizes of the sets from the standard input.

Solution

```
import random
even = set()
odd = set()
n = int(input("n = "))
m = int(input("m = "))
while len(even) < n:
    num = random.randint(1, 20)
    if num % 2 == 0:
        even.add(num)
while len(odd) < m:
    num = random.randint(1, 20)
    if num % 2 == 1:
        odd.add(num)
print(even)
print(odd)
print("Union =", even.union(odd))
```

Dictionary

- Dictionaries are used to store data values in **key:value** pairs.
- In Python, the easiest way to create a dictionary, or **dict**-type container, is to list the **key:value** pairs between curly brackets, with a colon between them.
  This is the most common form.
- The curly brackets can be empty, in which case an empty dictionary is created, which can be modified later.
- **Note**: You can not create an empty set with curly brackets. Because there would be no difference between the empty set and the empty dictionary.
- If there is at least one element, it is clear which one is created: if there is / are colons, then it is a dictionary, if there are no colons, then it is a set.

Dictionary

Create and print a dictionary:

```
shoppinglist = {
    "apple": 2,
    "pear": 1,
    "peach": 5,
}
print(shoppinglist)
```

Result:

```
{'apple': 2, 'pear': 1, 'peach': 5}
```

Dictionary

- You can create an empty dictionary with the **dict()** constructor call, or simply with an empty pair of curly brackets: {}.
- Each element is added and accessed using the indexing operator.

```
shoppinglist = dict()      # or {}
shoppinglist["apple"] = 2
shoppinglist["pear"] = 1
shoppinglist["peach"] = 5
print(shoppinglist)
```

Dictionary

- A third option is to specify a list of items that make up the contents of the dictionary.
- In this case, each element of the dictionary must be a tuple containing the key-value pairs.

```
shoppinglist = dict([
    ("apple", 2),
    ("pear", 1),
    ("peach", 5),
])
print(shoppinglist)
```

Access dictionary items

```
print(shoppinglist)
 #{'apple': 2, 'pear': 1, 'peach': 5}
print(shoppinglist["apple"])  # 2
print("apple" in shoppinglist)  # True
del shoppinglist["apple"]
shoppinglist["plum"] = 7
print(shoppinglist)
#{'pear': 1, 'peach': 5, 'plum': 7}
try:
    print(shoppinglist["papaja"])
except KeyError as e:
    print("Not in it!")  # Not include
```

Access dictionary items

```
for key in shoppinglist:  #keys
    print(key)

for value in shoppinglist.values():  #values
    print(value)

print("Sum:", sum(shoppinglist.values()))

for key, value in shoppinglist.items():#key-values
    print(f"{key}: {value} kg")
```

Access dictionary items

- If you only want to know the values, the keys are irrelevant, you can use **.values().** This returns only the values in the dictionary in a list.
- The sum() function returns the sum of the values.
- The most convenient way to work with the dict when we can see both the keys and the values in the same time.
- This can be achieved by the **.items()** function.
- Then the iteration will return tuple items, which are the items of the dictionary (key, value).

Exercise

- Write a program that gets words until the empty string, and creates a statistic of the input words, i.e. how many times was each word entered.

Solution

```
occurrence = {}

word = input()
while word != "":
    if word in occurrence:
        occurrence[word] += 1
    else:
        occurrence[word] = 1
    word = input()

for word, item in occurrence.items():
    print(f"{word}: {item} item")
```

Monkey's habit

- https://progcont.hu/progcont/100356/?pid=201536

Monkey's habit

- Everyone knows that the monkeys' favorite food is the delicious sweet banana. It's no wonder that every monkey wants to collect most of these tasty morsels for himself.
- Write a program that reads the name of one monkey per line from the standard input lines until the end-of-file (EOF) and the number of bananas it has collected, in the following format:
- `monkey_name;bananas_number`
- The program writes to the standard output the names of the monkeys and the number of bananas they have collected, in lexicographic ascending order by monkey name, in the format given in the example output!

Monkey's habit

- Sample Input
  - Jambo;10
  - Kongo;5
  - Charlie;12
  - Jambo;10
  - Koko;14
  - Kongo;10
  - Jambo;5
  - Charlie;8

- Output for Sample Input
  - Charlie: 20
  - Jambo: 25
  - Koko: 14
  - Kongo: 15

Solution

```python
import sys
monkey={}
for line in sys.stdin:
    data = line.split(";")
    if data[0] in monkey:
        monkey[data[0]] += int(data[1])
    else:
        monkey[data[0]] = int(data[1])

for key in sorted(monkey):
    print ("{0}: {1}".format(key, monkey[key]))
```

Lambda functions

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.

Syntax:

- lambda *arguments* : *expression*
- The expression is executed and the result is returned.

```
print((lambda x, y, z: (x + y + z) * 10)(1, 2, 3))
60

lista = ["Jabba", "Han", "Luke"]
print(sorted(lista, key = lambda item: -len(item)))
['Jabba', 'Luke', 'Han']
```

Festival season

- https://progcont.hu/progcont/100357/?pid=201540

Festival season

- Concerts are the main source of income for popular music bands, which go to festivals all over the country to entertain their fans.
- Write a program that reads data from standard input to end-of-file (EOF) for various festivals.
- Each line contains the name of a festival and the names of the bands performing at it, in the following format:

- `festival_name:band_name[,band_name]...`

- The `festival_name` and the `band_name` are always one string, containing neither a colon nor a comma.
- Your program will write to the standard output the names of the bands in lexicographical ascending order, and the number of festivals they have participated in, in the form given in the example output!

Festival season

- Sample Input
  - `Sziget fesztival:Punnany Massif,Tankcsapda,Kowalsky meg a vega`
  - `Campus fesztival:Tankcsapda,Fish!,Anna and the Barbies`
  - `SZIN:Punnany Massif,Kowalsky meg a vega,Tankcsapda`
  - `Volt:Anna and the Barbies,Tankcsapda,Kowalsky meg a vega`

- Output for Sample Input
  - `Anna and the Barbies: 2`
  - `Fish!: 1`
  - `Kowalksy meg a vega: 3`
  - `Punnany Massif: 2`
  - `Tankcsapda: 4`

Solution

```
import sys
bands = {}
for line in sys.stdin:
    data = line.split(":")
    names = data[1].strip().split(",")
    for i in range(0,len(names)):
        if names[i] in bands:
            bands[names[i]] += 1
        else:
            bands[names[i]] = 1
for key in sorted(bands):
    print ("{0}: {1}".format(key, bands[key]))
```

Homework 1 – Club loyalty

- https://progcont.hu/progcont/100360/?pid=201553

Club loyalty

- Some athletes are loyal to a single club throughout their entire sporting careers. Others are not like that: they change clubs from time to time to advance their careers or ensure their financial well-being.
- Write a program that reads an integer from the standard input in the first line, and then reads the data of sports clubs from the subsequent lines, up to the end of the file (EOF). Each line contains the names of athletes who have ever been associated with the respective sports club in the following format:
- `club_name:athlete_name[,athlete_name]...`
- The program should write to the standard output the names of athletes who have been associated with more clubs than the specified integer! If there are multiple such athletes, order them based on the number of clubs they have been part of.
- The more clubs someone has been associated with, the higher they should appear on the list!
- If it is still impossible to decide between two or more athletes based on this criterion, write their names in lexicographically increasing order, as shown in the sample output.

Club loyalty

- Sample Input
  - 2
  - Golden Team:Grosics,Buzanszky,Lorant,Lantos,Bozsik,Zakarias,Budai,Kocsis,Hidegkuti,Puskas,Czibor
  - Mighty Magyars:Buzanszky,Lantos,Zakarias,Hidegkuti
  - Marvellous Magyars:Grosics,Buzanszky,Bozsik,Hidegkuti,Puskas
  - Magnificent Magyars:Lorant,Budai,Kocsis,Hidegkuti,Puskas
  - Magical Magyars:Buzanszky,Lantos,Hidegkuti,Puskas
- Output for Sample Input
  - Hidegkuti: 5
  - Buzanszky: 4
  - Puskas: 4
  - Lantos: 3

Homework 2
Every day an apple keeps the doctor away

- https://progcont.hu/progcont/100358/?pid=201544

Homework 2
Every day an apple keeps the doctor away

- We have heard the phrase in the title many times: fruit is an essential part of a healthy diet.
- Write a program that reads, line by line from the standard input, the names of fruits and the names of children who like that fruit. The structure of the lines is as follows:
- `fruit_name:child_name[,child_name]...`

- The names of fruits and children are unique; there are no two children with the same name who like the same fruit.
- Your program should write to the standard output the names of children in lexicographically increasing order, along with the number of different fruits each child likes, in the format shown in the sample output.

Homework 2
Every day an apple keeps the doctor away

Sample Input

```
apple:Pete,Kate
plum:John
banana:Kate,John,Charles
pineapple:Hannah,Anna,Penny
apricot:Fanny,Pete
```

Output for Sample Input

```
Anna: 1
Charles: 1
Fanny: 1
Hannah: 1
John: 2
Kate: 2
Penny: 1
Pete: 2
```

Homework 3
New curriculum model

- https://progcont.hu/progcont/100278/?pid=201288

Homework 3
New curriculum model

- Write a program that reads an integer (N) from the standard input in the first line. In the following N lines of input, there are subject codes, names, and credit values, one per line, in the following format:
- `Subject_code:subject_name:credit_number`
- The subject_code and subject_name are strings, and credit_number is a positive integer.
- The subject_code is always in the format "INBMM-xx-yy," where xx represents the recommended semester number, and yy is the unique identifier for the subject.
- The program should write to the standard output the names of the subjects sorted in ascending order based on the recommended semesters. If multiple subjects have the same recommended semester, list them in increasing order of their unique identifiers on the output!

Homework 3
New curriculum model

- Sample Input:

```
5
INBMM-02-07:Data Structures and Algorithms:6
INBMM-01-01:Algorithms and Fundamentals of Programming:2
INBMM-01-02:Electronics:6
INBMM-03-14:Fundamentals of Economics:6
INBMM-02-08:Mathematics for Engineers 2:6
```

Output for Sample Input:

```
Algorithms and Fundamentals of Programming
Data Structures and Algorithms
Electronics
Fundamentals of Economics
Mathematics for Engineers 2
```