**מבוא מורחב למעדי המחשב - תרגיל בית 1**

1. א. 8

ב. את סכום ספרותיו

ג. הפעולה היא int(). הבעיה היא שהפעולה לא יודעת להפוך ביטוי אריתמטי למשתנה מסוג integer.

ד. פונקציה זו מקבלת מחרוזת שהיא ביטוי אריתמטי ומחזירה את התוצאה הסופית בתור float/int. מודפס 470.

ה. הפונקציה מגיעה לספרה האחרונה, מחלקת אותה ללא שארית בעשר ומתקבל 0. 0 חלקי 10 שוב נותן 0 ונוצרת לולאה אינסופית כי התנאי תמיד מתקיים.

>>> '@'.isalnum()  
False  
>>> '&'.isalnum()  
False

1. א. isalnum() מקבלת מחרוזת ומחזירה True אם כולה מורכבת מאותיות אלפבית ומספרים. אחרת, היא מחזירה False.

ב. split() פועלת על string1 ומקבלת string2. הפעולה מפצלת את string1 לחלקים, לפי המיקומים של string2 בתוכו. היא לוקחת את כל החלקים שמופרדים על ידי string2 ומחזירה רשימה שמורכבת מחלקים אלו. אם היא מקבלת גם מספר n, היא מחלקת עד הפעם ה-n בה string2 מופיע.

<<<"Intro2ComputerScience".split("c")  
['Intro2ComputerS', 'ien', 'e']  
<<<"Intro2ComputerScience".split("c", 1)  
['Intro2ComputerS', 'ience']

ג. replace() פועלת על string1, מקבלת string2 ו-string3 ומחזירה string חדש שמורכב כך: כל פעם ש-string2 מופיע ב-string1 היא מחליפה אותו ב-string3. היא יכולה גם לקבל מספר n, ואז היא פועלת עד הפעם ה-n בה string2 מופיע.

>>> 'asdasdasd'.replace('as','@',1)  
'@dasdasd'

ד. join() פועלת על string1 ומקבלת string2 או רשימה. היא מחזירה string חדש שמכיל את איברי string2 או הרשימה מופרדים על ידי string1.

>>> '+'.join("asdasd")  
'a+s+d+a+s+d'  
>>> HA=["aaaaaa","b","c"]  
>>> '+'.join(HA)  
'aaaaaa+b+c'

ה. count() פועלת על string1 ומקבלת string2. הפונקציה מחזירה את מספר הפעמים ש-string2 מופיע ב-string1. היא גם יכולה לקלוט שני מספרים ואז היא סופרת רק בין התווים שממוקמים במקומות אלו.

>>> '123123123123'.count('2',4,7)  
1

ו. find() פועלת על string1 ומקבלת string2. הפונקציה מחזירה את המיקום של ההופעה הראשונה של string2 בתוך string1. אם הוא לא נמצא, היא מחזירה -1. ניתן גם להכניס מספרים ואז הבדיקה נעשית בין התווים הממוקמים במקומות אלו.

>>> 'zxcAzxczxcAzxc'.find('A',2,5)  
3  
>>> 'zxcABzxc'.find('ABC')  
-1

>>> 'zxcABzxc'.index('ABC')  
Traceback (most recent call last):  
 File "<pyshell#130>", line 1, in <module>  
 'zxcABzxc'.index('ABC')  
ValueError: substring not found

Index() זהה ל-find(), פרט לכך שאם string2 לא נמצא ב-string1, היא מחזירה הודעת שגיאה.

>>> a  
['a', 'b', 'c']  
>>> a.append('d')  
>>> a  
['a', 'b', 'c', 'd']

>>> a=['a','b','c']  
>>> a.extend([5,'C','W'])  
>>> a  
['a', 'b', 'c', 5, 'C', 'W']

**List**

ז. append() מוסיפה איבר לסוף רשימה.

extend() מוסיפה לרשימה קיימת איברי רשימה אחרת.

pop() מוחקת מרשימה את האיבר האחרון ומדפיסה אותו. היא יכולה גם לקבל אינדקס ואז היא פועלת על האיבר במיקום זה.

>>> a  
['a', 'b', 'c', 5, 'C', 'W']  
>>> a.pop(2)  
'c'  
>>> a  
['a', 'b', 5, 'C', 'W']  
>>> a.pop()  
'W'  
>>> a  
['a', 'b', 5, 'C']

>>> a  
['a', 'b', 5, 'C']  
>>> a.remove('b')  
>>> a  
['a', 5, 'C']

>>> a  
['a', 5, 'C']  
>>> a.insert(2,'WWW')  
>>> a  
['a', 5, 'WWW', 'C']

remove() מקבלת איבר ומוחקת מהרשימה עליה היא פועלת את אותו איבר.

insert() מקבלת אינדקס ואיבר כלשהו. היא מכניסה לרשימה את האיבר במיקום של האינדקס.

sort() ממיינת איברי רשימה לפי אלפבית או לפי גודל מספר. אם ישנם גם מספרים וגם אותיות היא מחזירה הודעת שגיאה.

>>> a  
[1, 6345435, 34, 77777, 123]  
>>> a.sort()  
>>> a  
[1, 34, 123, 77777, 6345435]  
>>> a=['dfggg','asd','bzxczxc','cWW']  
>>> a.sort()  
>>> a  
['asd', 'bzxczxc', 'cWW', 'dfggg']

>>> a  
[5, 123, 5, 66, 5]  
>>> a.count(5)  
3

>>> a=['a','b','c','a']  
>>> a.index('a')  
0

count() מקבלת איבר ומחזירה כמה פעמים הוא מופיע ברשימה.

index() מקבלת איבר מסוים ומחזירה את האינדקס של ההופעה הראשונה של אותו איבר ברשימה. היא יכולה גם לקבל אינדקסים ואז היא מבצעת את הבדיקה ביניהם.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 2\*\*200 | 2\*\*400 | 2\*\*800 | 2\*\*1600 |
| פתרון 1 | 3.154909876008241e-05 sec | 7.146060923970473e-05 sec | 0.00020830007374126697 sec | Running time: 0.0006423852639101117 sec |
| פתרון 2 | 9.882850214001719e-06 sec | 1.4444165697387127e-05 sec | 2.8508221771158802e-05 sec | 5.397556655339399e-05 sec |
| פתרון 3 | 5.32153473061631e-06 sec | 6.841973225078112e-06 sec | 1.2543617579309873e-05 sec | 2.6987783276697e-05 sec |

4. א.

בפתרון 1 ניתן לראות שהיחס בין המדידות גדל. יחס של 7/3 בין המדידה השנייה לראשונה, יחס של כמעט פי 3 בין המדידה השלישית לשנייה, ויחס של יותר מפי 3 בין המדידה השלישית לרביעית.  
בפתרון 2 ניתן לראות שהיחס נע בסביבות פי 2 יותר זמן בין כל שתי מדידות עוקבות. פתרון 2 יותר מהיר מפתרון 1.  
פתרון 3 הכי מהיר, פי 2 בערך מפתרון 2, וכמובן שהיחס בין כל שתי מדידות עוקבות דומה לזה של פתרון 2.

|  |  |  |
| --- | --- | --- |
|  | 10\*\*1000 | 10\*\*1000-1 |
| פתרון 1 | 0.0024418242221056555 sec | 0.002422438631301267 sec |
| פתרון 2 | 0.00018397305782987813 sec | 9.616773477470903e-05 sec |
| פתרון 3 | 8.894565192601546e-05 sec | 8.970587117324637e-05 sec |

ג.

פתרון 1- הזמנים כמעט זהים, ניתן להסיק מכך שזמן הביצוע של הפעולה cnt+=1, שמתקיימת רק אם הספרה שנבדקת היא 0, הוא זניח ביחס לשאר הפעולות.  
פתרון 2- זמן הבדיקה של מספר שמכיל רק אפסים פרט לספרה הראשונה הינו כפול מזמן הבדיקה של מספר באותו אורך ללא אפסים. כאן זמן הביצוע של הפעולה cnt+=1 מהווה חלק גדול יותר מזמן הביצוע של כל התכנית ביחס לפתרון 1.  
פתרון 3- כאן הזמנים כמעט זהים. ניתן להסיק כי האלגוריתם של הפעולה בנוי כך שאין השפעה גדולה של הקלט על זמן הפעולה.

ד. זוהי לולאה שרצה 2^100 פעמים לכן זה יקח הרבה זמן יחסית. ללולאה בפתרון 2 לקח 10 מיקרו-שניות לרוץ על המספר 200^2, שיש לו בערך 60 ספרות. להערכתי הזמן שיקח ללולאה זו הוא באותו יחס, כלומר, 60 / 100^2 \* 10 מיקרו שניות.

הלולאה בסעיף א' רצה על כל ספרה, כלומר כאורך המספר. בסעיף זה הלולאה רצה מספר הפעמים השווה לגודלו של המספר, ולכן בהכרח זמן הריצה גדול בהרבה.

6.

>>>

Please enter a positive integer: 7\*\*444444

The maximal length is 16

Sequence starts at 232899

Sequence is 1775913957791171

>>>