ՀՀ կրթության, գիտության, մշակույթի և սպորտի նախարարություն

Հայաստանի Ազգային Պոլիտեխնիկական Համալսարան (ՀԻՄնադրամ)

Ալգորիթմական լեզուների և ծրագրավորման ամբիոն

ՏՀՏԷ ինստիտուտի ԱԼ և Ծ ամբիոն_
ՀՏՏ 924-1 ակադեմիական խումբ_
Մասնագիտություն Տեղեկատվական Տեխնոլոգիաներ

ԿՈՒՐՍԱՅԻՆ ԱՇԽԱՏԱՆՔԻ ԱՌԱՋԱԴՐԱՆՔ

_____
(առարկայի անվանումը)

_____ Հովհաննիսյան Սերգեյ Օնիկի_____
(ուսանողի ազգանուն, անուն, հայրանուն)

1. Աշխատանքի թեման՝ Բուլյան ֆունկցիաներ
2. Աշխատանքի նախնական տվյալները _____
3. Հաշվեբացատրագրի բովանդակությունը

3.1. Խնդրի դրվածքը.

3.2. Ալգորիթմներ, դրանց նկարագրությունը

3.3 Ծրագրի օգտագործման տեխնոլոգիան

4. Գրաֆիկական մասի ծավալը
_____

5. Կատարման ժամանակացույցը
_____

6 Աշխատանքի ղեկավար
_____
(գիտական աստիճան, տարակարգ, ստորագրություն, ամսաթիվ)

7. Ամբիոնի վարիչ
_____
(գիտական աստիճան, տարակարգ, ստորագրություն, ամսաթիվ)

8. Ուսանող _____
(ամսաթիվ, ոսանողի ստորագրություն)

ԵՐԵՎԱՆ 2021

ՀՀ կրթության, գիտության, մշակույթի և սպորտի նախարարություն

Հայաստանի Ազգային Պոլիտեխնիկական Համալսարան (ՀԻմնադրամ)

Ալգորիթմական լեզուների և ծրագրավորման ամբիոն

ՀԱՇՎԵԲԱՑԱՏՐԱԳԻՐ

առարկայի կուրսային աշխատանքի

Թեմա՝ _____

_____

_____

_____

_____

_____

Ուսանող

_____

(ազգանուն, անուն, հայրանուն, ամսաթիվ, ստորագրություն)

Ղեկավար՝ _____ Պողոսյան _____

(Ա.Հ.Ա., ստորագրություն, ամսաթիվ)

Ամբիոնի վարիչ

_____

(գիտական աստիճան, տարակարգ, ստորագրություն, ամսաթիվ)

Հանձնաժողովի անդամներ _____

(գիտական աստիճան, տարակարգ, ստորագրություն, ամսաթիվ)

_____

(գիտական աստիճան, տարակարգ, ստորագրություն, ամսաթիվ)

_____

(գիտական աստիճան, տարակարգ, ստորագրություն, ամսաթիվ)

ԵՐԵՎԱՆ 2021

Բովանդակություն

Ներածություն

Տվյալների տեսակները Python - ում

Python-ի յուրաքանչյուր արժեք ունի տվյալների տեսակ: Քանի որ Python ծրագրավորման մեջ ամեն ինչ օբյեկտ է, տվյալների տեսակները իրականում դասեր են, իսկ փոփոխականները՝ այս դասերի օրինակ (օբյեկտ):

Python-ում տվյալների տարբեր տեսակներ կան: Կարևոր տեսակներից մի քանիսը թվարկված են ստորև:

Numbers

Ամբողջ թվերը, լողացող կետով թվերը և կոմպլեքս թվերը պատկանում են Python **Numbers** կատեգորիային: Python-ում դրանք սահմանվում են որպես int, float և կոմպլեքս դասեր:

Մենք կարող ենք օգտագործել type() ֆունկցիան՝ իմանալու համար, թե որ դասին է պատկանում փոփոխականը կամ արժեքը: Նմանապես, isinstance() ֆունկցիան օգտագործվում է ստուգելու համար, թե արդյոք օբյեկտը պատկանում է որոշակի դասի:

Օրինակ՝

```
a = 5
print(a, "is of type", type(a))

a = 2.0
print(a, "is of type", type(a))

a = 1+2j
print(a, "is complex number?", isinstance(1+2j,complex))
```

Print

```
5 is of type <class 'int'>

2.0 is of type <class 'float'>

(1+2j) is complex number? True
```

List

Ցուցակը էլեմենտների հաջորդականություն է: Այն Python-ում ամենաշատ օգտագործվող տվյալների տեսակներից մեկն է և շատ ճկուն է: **List** - ի բոլոր տարրերը պարտադիր չէ, որ լինեն նույն տեսակի:

Ցուցակ հայտարարելը բավականին պարզ է: Ստորակետերով առանձնացված կետերը փակցված են [ ] փակագծերում:

Մենք կարող ենք օգտագործել կտրատման օպերատորը [ ]՝ ցանկից որևէ տարր կամ տարրերի տիրույթ հանելու համար: Python-ում ինդեքսը սկսվում է 0-ից:
Օրինակ՝

```
a = [5,10,15,20,25,30,35,40]

# a[2] = 15
print("a[2] = ", a[2])

# a[0:3] = [5, 10, 15]
print("a[0:3] = ", a[0:3])

# a[5:] = [30, 35, 40]
print("a[5:] = ", a[5:])
```

Ցանկերը **mutable** (փոփոխական) են, այսինքն՝ ցանկի տարրերի արժեքը կարող է փոփոխվել:

Tuple

Tuple-ը էլեմենտների հաջորդականություն է, որը նույնն է, ինչ **List** - ը: Միակ տարբերությունն այն է, որ **tuple** - ները անփոփոխ են: Մի անգամ ստեղծվող **tuple** - ները չեն կարող փոփոխվել:

Tuple-ներն օգտագործվում են տվյալների գրավոր պաշտպանության համար և սովորաբար ավելի արագ են, քան ցուցակները, քանի որ դրանք չեն կարող դինամիկ կերպով փոխվել:

Այն սահմանվում է փակագծերում () որտեղ էլեմենտները առանձնացված են ստորակետերով:

Օրինակ՝

```
t = (5,'program', 1+3j)

# t[1] = 'program'
print("t[1] = ", t[1])

# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])

# Generates error
# Tuples are immutable
t[0] = 10
```

Տպում

```
t[1] =  program
t[0:3] =  (5, 'program', (1+3j))
Traceback (most recent call last):
  File "test.py", line 11, in <module>
    t[0] = 10
TypeError: 'tuple' object does not support item assignment
```

String

String - երը Յունիկոդի նիշերի հաջորդականություն է: **String** - երը ներկայացնելու համար մենք կարող ենք օգտագործել միայնակ չակերտներ կամ կրկնակի չակերտներ: Բազմատող տողերը կարող են նշանակվել՝ օգտագործելով եռակի չակերտներ՝ ''' կամ """

Օրինակ՝

```
s = "This is a string"
print(s)
s = '''A multiline
string'''
print(s)
```

Ճիշտ այնպես, ինչպես List - երը և Tuple - ները, կտրատման օպերատորը [ ] կարող է օգտագործվել տողերի հետ: **String** - նեը, սակայն, անփոփոխ են:

Օրինակ՝

```
s = 'Hello world!'

# s[4] = 'o'
print("s[4] = ", s[4])

# s[6:11] = 'world'
print("s[6:11] = ", s[6:11])

# Generates error
# Strings are immutable in Python
s[5] ='d'
```

Set

Set-ը եզակի էլեմենտների հավաքածու է: Set-ը սահմանվում է {} փակագծերի ներսում ստորակետերով բաժանված արժեքներով: Կոմպլեկտի իրերը կարգված չեն:

Օրինակ՝

```
a = {5,2,3,1,4}

# printing set variable
print("a = ", a)

# data type of variable a
print(type(a))
```

Տպում

```
a =  {1, 2, 3, 4, 5}
<class 'set'>
```

Մենք կարող ենք կատարել մի շարք գործողություններ, ինչպիսիք են միավորումը, հատումը երկու բազմությունների վրա: Set - երը ունեն յուրահատուկ արժեքներ: Նրանք վերացնում են կրկնօրինակները:

Քանի որ Set-ը չկարգված հավաքածու է, ինդեքսավորումն իմաստ չունի: Հետևաբար, կտրող օպերատորը **[ ]** չի աշխատում:

Օրինակ՝

```
>>> a = {1,2,3}
>>> a[1]
Traceback (most recent call last):
  File "<string>", line 301, in runcode
  File "<interactive input>", line 1, in <module>
TypeError: 'set' object does not support indexing
```

Dictionary

Dictionary - ին բանալի-արժեք զույգերի չկարգված հավաքածու է:

Այն սովորաբար օգտագործվում է, երբ մենք ունենք հսկայական քանակությամբ տվյալներ: **Dictionary - ները** օպտիմիզացված են տվյալների որոնման համար: Արժեքը ստանալու համար մենք պետք է իմանանք բանալին:

Python-ում **Dictionary - ները** սահմանվում են {} փակագծերում, որոնցից յուրաքանչյուրը զույգ է՝ **key:value** ձևով: Բանալին և արժեքը կարող են լինել ցանկացած տեսակի:

Օրինակ՝

```
>>> d = {1:'value','key':2}
>>> type(d)
<class 'dict'>
```

Մենք օգտագործում ենք բանալին համապատասխան արժեքը ստանալու համար: Բայց ոչ հակառակը:

Օրինակ՝

```
d = {1:'value','key':2}
print(type(d))

print("d[1] = ", d[1])

print("d['key'] = ", d['key'])

# Generates error
print("d[2] = ", d[2])
```

Տպում

```
<class 'dict'>
d[1] =  value
d['key'] =  2
Traceback (most recent call last):
  File "<string>", line 9, in <module>
KeyError: 2
```

Տվյալների տեսակների փոխակերպում

Մենք կարող ենք փոխակերպել տվյալների տարբեր տեսակների միջև՝ օգտագործելով տարբեր տեսակի փոխակերպման գործառույթներ, ինչպիսիք են int(), float(), str() և այլն:

Օրինակ՝

```
>>> float(5)
5.0
```

float-ից **int**-ի փոխակերպումը կկտրի արժեքը (այն մոտեցնում է զրոյին):

Օրինակ՝

```
>>> int(10.6)
10
>>> int(-10.6)
-10
```

Տվյալների վիզուալիզացիա

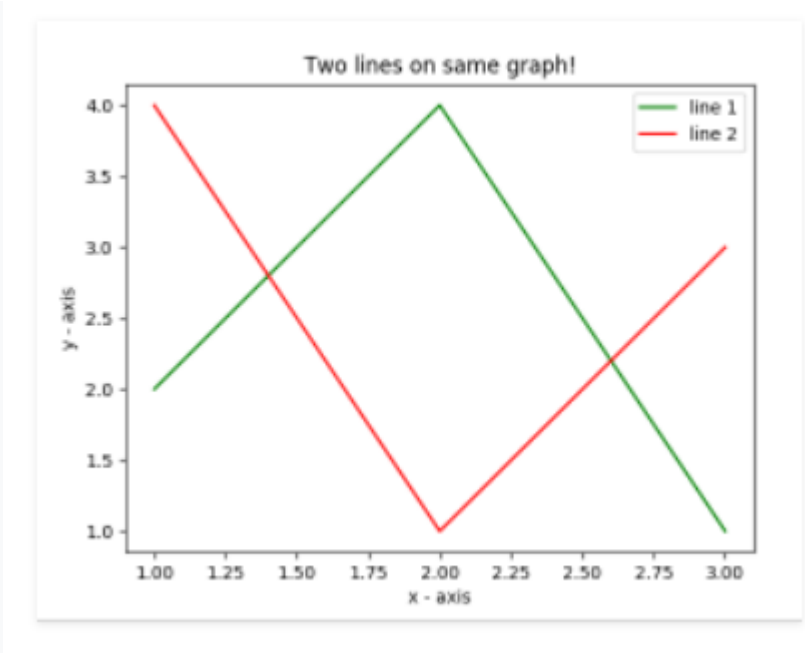
Այս շարքը ձեզ կներկայացնի Matplotlib-ի միջոցով python-ում գծապատկերներ կազմելը, որը, հավանաբար, Python-ի համար ամենահայտնի գրաֆիկական և տվյալների արտացոլման գրադարանն է:

Տեղադրում

Matplotlib-ը տեղադրելու ամենահեշտ ձևը pip-ն օգտագործելն է: Մուտքագրեք հետևյալ հրամանը տերմինալում:

Կոդն ինքնաբացատրելի է թվում: Հետևյալ քայլերն իրականացվել են.

- Սահմանեք x առանցքի և համապատասխան y առանցքի արժեքները որպես ցուցակներ:
- Գրեք դրանք կտավի վրա՝ օգտագործելով .plot() ֆունկցիան:
- Անվանեք x առանցքին և y առանցքին՝ օգտագործելով .xlabel() և .ylabel() ֆունկցիաները:
- Վերնագիր տվեք ձեր հողամասին՝ օգտագործելով .title() ֆունկցիան:
- Վերջապես, ձեր սյուժեն դիտելու համար մենք օգտագործում ենք .show() ֆունկցիան:



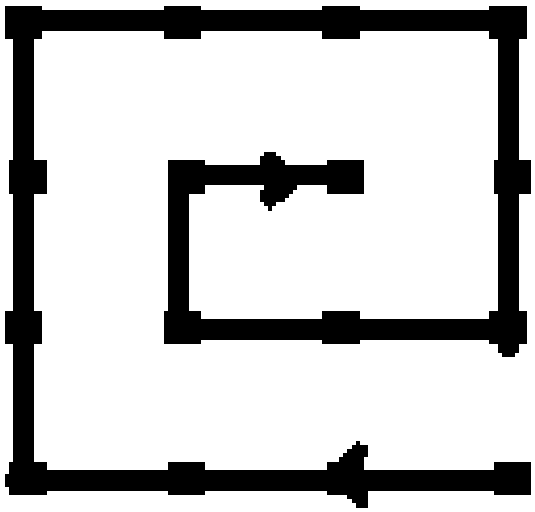
Այստեղ մենք գծում ենք երկու տող նույն գրաֆիկի վրա: Մենք դրանք տարբերում ենք՝ տալով անուն(պիտակ), որը փոխանցվում է որպես .plot() ֆունկցիայի արգումենտ:

Փոքր ուղղանկյուն տուփը, որը տեղեկատվություն է տալիս գծի տեսակի և դրա գույնի մասին, կոչվում է լեգենդ: Մենք կարող ենք լեգենդ ավելացնել մեր սյուժեին՝ օգտագործելով .legend() ֆունկցիան:

Խնդրի դրվածքը

Կուրսային աշխատանքի խնդիրն է `

Արտատպել երկչափ զանգվածը ըստ տրված սխեմայի:



Ալգորիթմի նկարագրությունը

Ալգորիթմը կատարվում է **snail** անունով ֆունկցիայի օգնությամբ. ֆունկցիան վերադարձնում է վեկտոր այն իր մեջ պարունակում է բոլոր այն տարրերը որոնք վերցվել են մատրիցից տրվախ սխեմայով: Եկեք անցնենք ֆունկցիայի իրականացման տեղերով բաժանենք մասերի և մաս մաս նկարագրենք ֆունկցիայի կատարումը `

 # Ստեղծում ենք վեկտոր որտող պետք է գրանցենք մեր տարրերը

```
res = []
```

 # Այստեղ ստուգում ենք եթե մեր ֆունկցաին ստացել է դատարկ մարտիցա վերադարձնում ենք ստեղծած դատարկ վեկտորը

```
if len(matrix) == 0:
return res
```

 # Ստեղծում ենք փոփոխականները որտեղ գրանցում ենք մատրիցի սկզբնական վերջնական տողերի և սյուների համարները – ինդեքսները

```
row_begin = 0 -------> առաջին տող
row_end = len(matrix) – 1  -------> վերջին տող
col_begin = 0 -----------> առաջին սյուն
col_end = len(matrix[0]) – 1  -----> վերջին սյուն
```

 # Ստեղծում ենք while ցիկլ որի կկատարվի քանի դեռ մեր տողի սկիզբը փոքր է կամ հավասար տողի վերջին և սյան սկիզբը փոքր է կամ հավասար սյան վերջի ինդեքսից.

```
while row_begin <= row_end and col_begin <= col_end:
```

 # Սկսում ենք վերջին սյան վերջին տողից գնում ենք վերջին տողի վրայով տեպի առաջին սյան վրա որից հետո վերջի տողի ինդեքսը փոքրացնում ենք 1 ով

```
for i in range(col_end, col_begin-1, -1):
        res.append(matrix[row_end][i])
 row_end -= 1
```

 # Որից հետո գնում ենք գնում ենք առաջին տողի վրա առաջին սյունակով բարձրանալով հետո առաջին սյունակի ինդեքսը մեծացնում ենք մեկով

```
for i in range(row_end, row_begin-1,-1):
   res.append(matrix[i][col_begin])
col_begin += 1
```

Որից հետո գնում ենք գնում ենք վերջին սյան վրա առաջին տողով. վերջին տողի ինդեքսը մեծացնում ենք մեկով

```
        for i in range(col_begin, col_end+1):
            res.append(matrix[row_begin][i])
        row_begin += 1
```

Որից հետո գնում ենք գնում ենք նախավերջին տողի վրա վերջին սյունակով. վերջին սյունակի ինդեքսը մոքրացնում ենք մեկով

```
      for i in range(row_begin, row_end+1):
          res.append(matrix[i][col_end])
      col_end -= 1
```

 # Կրկնում ենք քանի դեռ while ցիկլի պայմանը կեղծ չէ

Որից հետո վերարձնում ենք ստացված վեկտորը

```
      return res
```

Ծրագրի կոդը

```
def snail(matrix):
    res = []
    if len(matrix) == 0:
        return res
    row_begin = 0
    row_end = len(matrix) - 1
    col_begin = 0
    col_end = len(matrix[0]) - 1

    while row_begin <= row_end and col_begin <= col_end:

        for i in range(col_end, col_begin-1, -1):
            res.append(matrix[row_end][i])
        row_end -= 1

        for i in range(row_end, row_begin-1,-1):
            res.append(matrix[i][col_begin])
        col_begin += 1



        for i in range(col_begin, col_end+1):
            res.append(matrix[row_begin][i])
        row_begin += 1

        for i in range(row_begin, row_end+1):
            res.append(matrix[i][col_end])
        col_end -= 1

    return res

matrix = [
  [1,  2 , 3,  4],
  [6,  7 , 8 , 9],
  [11, 12, 13, 14],
  [16, 17, 18, 19]
]
print(snail(matrix))
```

Ծրագրի աշխատանքի արդյունքը

Input

```
[1,  2 , 3,  4]
[6,  7 , 8 , 9]
[11, 12, 13, 14]
[16, 17, 18, 19]
```

Output

[19, 18, 17, 16, 11, 6, 1, 2, 3, 4, 9, 14, 13, 12, 7, 8]

Եզրակացություն

Առարկան ուսումնասիրելու ժամանակ կուրսային աշխատանքը պատրաստել եմ օգտագործելով Python – ի List – երը հասկացա որ python լեզուն չի պահանջում նկարագրել թէ ինչ տիպի պետք է լինեն մեր տվյալները, որը չափազանց հեշտացնում է խնդիրների կատարման ձևը ու կրճատում ժամանակը:

Գրականություն

Օգտագործված գրականությունը `

1.https://ourcodeworld.com/articles/read/827/how-to-format-a-given-array-matrix-in-spiral-form-snail-or-clockwise-spiral-sorting-in-python
2.https://afteracademy.com/blog/spiral-order-traversal-of-a-matrix