Функциялар мен модульдер

Функциялар

Python-да функциясының құрылымы

```
Def func_name(param1, param2,. . .):
    операторлар
return қайтаратын_мәндер
```

мұндағы param1, param2,. . . параметрлер болып табылады. Параметр болып Python-дағы кез келген нысан, соның ішінде функция да болуы мүмкін. Параметрлерге әдепкі(по умолчанию) мәндер берілуі мүмкін, бұл жағдайда функция шақыруындағы параметр міндетті емес. `return` операторы немесе қайтару мәндері болмаса, функция нөлдік нысанды қайтарады.

Келесі функция $f(x)$ санының алғашқы екі туындысын ақырлы айырымдар арқылы есептейді:

```
def derivatives(f,x,h=0.0001): # h әдепкі мәні бар
    df =(f(x+h) - f(x-h))/(2.0*h)
    ddf =(f(x+h) - 2.0*f(x) + f(x-h))/h**2
    return df,ddf
```

Енді осы функцияны x = 0,5 кезінде arctan(x) екі туындысын анықтау үшін қолданайық:

```
from math import atan
df,ddf = derivatives(atan,0.5) # әдепкі h мәнін пайдаланады
print('Бірінші туынды =',df)
print('Екінші ретті туынды =',ddf)
```

Байқап тұрғандай, `derivatives` функциясына `atan` параметр ретінде берілетінін ескеріңіз. Бағдарламаның нәтижесі

```
Бірінші туынды = 0.7999999995730867
Екінші ретті туынды = -0.6399999918915711
```

Функция анықтамасындағы енгізу параметрлерінің саны ерікті түрде қалдырылуы мүмкін. Мысалы, келесі функция анықтамасында

```
def func(x1,x2,*x3)
```

x1 және x2 әдеттегі параметрлер болып табылады, олар *позициялық параметрлер* деп те аталады, ал x3 – *артық параметрлерді* қамтитын еркін ұзындықты кортежі. Бұл функцияны

```
def func(a,b,c,d,e)
```

түрінде шақыру параметрлер арасындағы келесі сәйкестікке әкеледі:

$$a \leftrightarrow x1, b \leftrightarrow x2, (c, d, e) \leftrightarrow x3$$

Позициялық параметрлер әрқашан артық параметрлердің алдында көрсетілуі керек.

Тізім сияқты өзгеретін нысан өзгертілетін функцияға берілсе, өзгерістер шақырушы бағдарламада да пайда болады. Келесі мысал көрейік:

```
def squares(a):
    for i in range(len(a)):
        a[i] = a[i]**2

a = [1, 2, 3, 4]
squares(a)
print(a) # 'a' енді 'a**2' бар
```

Нәтижесі

```
[1, 4, 9, 16]
```

Лямбда операторы

Егер функцияның өрнек формуламен берілсе, онда оны lambda операторымен анықтауға болады

```
func_name = lambda param1,param2,…:expression
```

Бірнеше операторларға рұқсат етілмейді. Мысалы:

```
>>> c = lambda x,y : x**2 + y**2
>>> print(c(3,4))
25
```

Модульдер

Пайдалы функцияларды модульдерде сақтау дұрыс тәжірибе. Модуль – бұл функциялар орналасқан жай ғана файл; модульдің аты файлдың аты болып табылады. Модульді бағдарламаға

```
from module_name import *
```

оператор арқылы жүктеуге болады.

Python әртүрлі тапсырмалар үшін функциялар мен әдістерді қамтитын көптеген модульдермен бірге келеді. Кейбір модульдер келесі екі бөлімде қысқаша сипатталған. Қосымша модульдер, соның ішінде графикалық пакеттер, интернеттен жүктеп алуға болады.

Математикалық модульдер

math модулі

Көптеген математикалық функциялар негізгі Python-ға салынбаған, бірақ математикалық модульді жүктеу арқылы қол жетімді болады. Модульдегі функцияларға қол жеткізудің үш жолы бар. Оператор

```
from math import *
```

`math` модульдегі барлық функция анықтамаларын ағымдағы функцияға немесе модульге жүктейді. Бұл әдісті пайдалану ұсынылмайды, себебі ол тек ысырапшылдық ғана емес, сонымен қатар басқа модульдерден жүктелген анықтамалармен қайшылықтарға әкелуі мүмкін. Мысалы, Python модульдерінде синус функциясының үш түрлі анықтамасы бар, `math, cmath` және `numpy`. Егер сіз осы модульдердің екі немесе одан да көбін жүктеген болсаңыз, sin(x) функция шақыруында қандай анықтаманың қолданылатыны белгісіз (ол соңғы жүктелген модульдегі анықтама).

Қауіпсіз, бірақ сенімсіз әдіс – таңдалған анықтамаларды келесі түрде жүктеу

```
from math import func1,func2,...
```

төмендегі мысалда көрсетілген:

```
>>> from math import log,sin
>>> print(log(sin(0.5)))
-0.735166686385
```

Алдымен модульді мәлімдемемен қол жетімді ету арқылы қайшылықтарды толығымен болдырмауға болады

```
import math
```

содан кейін модуль атауын префикс ретінде пайдалану арқылы модульдегі анықтамаларға қол жеткізу. Мысалы:

```
>>> import math
>>> print(math.log(math.sin(0.5)))
-0.735166686385
```

Сондай-ақ, модуль бүркеншік атпен қолжетімді болуы мүмкін. Мысалы, *math* модулін m бүркеншік атымен келесі командамен қол жетімді болуы мүмкін

```
import math as m
```

Енді *math* орнына m префиксі қолданылады:

```
>>> import math as m
>>> print(m.log(m.sin(0.5)))
-0.735166686385
```

Модульдің мазмұнын dir(*модуль*) шақыру арқылы басып шығаруға болады. *math* модульдегі функциялардың тізімін алу жолы:

```
>>> import math
>>> dir(math)
 ['__doc__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh',
'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist',
 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor',
 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
 'isfinite', 'isinf', 'isnan', 'isqrt', 'ldexp', 'lgamma', 'log',
 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow',
 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan',
 'tanh', 'tau', 'trunc']
```

Бұл функциялардың көпшілігі бағдарламашыларға таныс. Модуль екі тұрақтыны қамтитынын ескеріңіз: π және e.

cmath модулі

cmath модулі math модульдегі көптеген функцияларды қамтамасыз етеді, бірақ бұл функциялар комплекс сандарды қабылдайды. Модульдегі функциялар

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__',
 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atanh', 'cos', 'cosh',
 'e', 'exp', 'inf', 'infj', 'isclose', 'isfinite', 'isinf', 'isnan',
 'log', 'log10', 'nan', 'nanj', 'phase', 'pi', 'polar', 'rect',
 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau']
```

Мұнда комплекс сандармен мысалдары берілген:

```
>>> from cmath import sin
>>> x = 3.0 -4.5j
>>> y = 1.2 + 0.8j
>>> z = 0.8
>>> print(x/y)
(-2.56205313375e-016-3.75j)
>>> print(sin(x))
(6.35239299817+44.5526433649j)
>>> print(sin(z))
(0.7173560909+0j)
```

numpy модулі

Негізгі ақпарат

`numpy` модулі стандартты Python шығарылымының бөлігі емес. Жоғарыда айтылғандай, оны бөлек орнату керек (орнату өте оңай). Модуль тізімдерге ұқсас массив нысандарын ұсынады, бірақ модульде қамтылған көптеген функциялар арқылы түрлендіріп басқаруға болады. Массив өлшемі өзгермейді және бос элементтерге рұқсат етілмейді.

Numpy ішіндегі функциялардың толық жинағы толығымен басып шығару үшін тым ұзақ. Төмендегі тізім ең жиі қолданылатын функциялармен шектелген.

```
['complex', 'float', 'abs', 'append', arccos', 'arccosh',
'arcsin', 'arcsinh', 'arctan', 'arctan2', 'arctanh', 'argmax',
'argmin', 'cos', 'cosh', 'diag', 'diagonal', 'dot', 'e', 'exp',
'floor', 'identity', 'inner, 'inv', 'log', 'log10', 'max', 'min',
'ones', 'outer', 'pi', 'prod' 'sin', 'sinh', 'size', 'solve',
'sqrt', 'sum', 'tan', 'tanh', 'trace', 'transpose',
'vectorize','zeros']
```

Массив құру

Массивтерді бірнеше жолмен жасауға болады. Олардың бірі тізімді массивке айналдыру үшін `array` функциясын пайдалану болып табылады:

array(list, type)

Төменде өзгермелі нүкте элементтері бар 2×2 массивін құрудың екі мысалы келтірілген:

```
>>> from numpy import array
>>> a = array([[2.0, -1.0],[-1.0, 3.0]])
>>> print(a)
[[ 2. -1.]
[-1. 3.]]
>>> b = array([[2, -1],[-1, 3]],float)
>>> print(b)
[[ 2. -1.]
[-1. 3.]]
```

Басқа қол жетімді функциялар

zeros((dim1, dim2),type)

мұнда $dim1 \times dim2$ массивін жасайды және оны нөлдермен толтырады және

ones((dim1, dim2),type)

мұнда массивті бірліктермен толтырады. Екі жағдайда да әдепкі типі `float` өзгермелі нүкте болып табылады.

Соңында, тағы бір функцияны қарайық

arange(from,to,increment)

ол *range* диапазон функциясы сияқты жұмыс істейді, бірақ тізбекті емес массивті қайтарады. Мұнда массивтерді құру мысалдары берілген:

```
>>> from numpy import *
>>> print(arange(2,10,2))
[2 4 6 8]
```

```
>>> print(arange(2.0,10.0,2.0))
[ 2. 4. 6. 8.]
>>> print(zeros(3))
[ 0. 0. 0.]
>>> print(zeros((3),int))
[0 0 0]
>>> print(ones((2,2)))
[[ 1. 1.]
[ 1. 1.]]
```

Массив элементтеріне қол жеткізу және өзгерту

Егер `a` рангісі 2 массив болса, онда *i* жолындағы және *j* бағанындағы элементке `a[i,j]` қол жетімді болады, ал `a[i]` *i* жолына сілтеме жасайды. Массив элементтерін меншіктеу арқылы келесідей өзгертуге болады:

```
>>> from numpy import *
>>> a = zeros((3,3),int)
>>> print(a)
[[0 0 0]
[0 0 0]
[0 0 0]]
>>> a[0] = [2,3,2] # жолды өзгерту
>>> a[1,1] = 5 # элементті өзгерту
>>> a[2,0:2] = [8,-3] # жолддың бөлігін өзгерту
>>> print(a)
[[ 2 3 2]
[ 0 5 0]
[ 8 -3 0]]
```

Массивтердегі амалдар

Арифметикалық операторлар кортеждер мен тізімдерге қарағанда массивтерде басқаша жұмыс істейді — операция массивтің барлық элементтеріне таратылады; яғни операция массивтің әрбір элементіне қолданылады. Мысалдар:

```
>>> from numpy import array
>>> a = array([0.0, 4.0, 9.0, 16.0])
>>> print(a/16.0)
[ 0. 0.25 0.5625 1. ]
>>> print(a - 4.0)
[ -4. 0. 5. 12.]
```

numpy-де қол жетімді математикалық функциялар да келесідей таратылады:

```
>>> from numpy import array,sqrt,sin
>>> a = array([1.0, 4.0, 9.0, 16.0])
>>> print(sqrt(a))
[ 1. 2. 3. 4.]
>>> print(sin(a))
[ 0.84147098 -0.7568025 0.41211849 -0.28790332]
```

`math` модульден импортталған функциялар, әрине, жеке элементтерде жұмыс істейді, бірақ массивтің өзінде емес. Келесі мысал:

```
>>> from numpy import array
>>> from math import sqrt
>>> a = array([1.0, 4.0, 9.0, 16.0])
>>> print(sqrt(a[1]))
```

```
2.0
>>> print(sqrt(a))
TypeError                                 Traceback (most recent call last)
<ipython-input-45-29539b5bdc2d> in <module>
----> 1 print(sqrt(a))

TypeError: only size-1 arrays can be converted to Python scalars
```

Массив функциялары

Numpy-де массив операцияларын және басқа да пайдалы тапсырмаларды орындайтын көптеген функциялар бар. Міне, бірнеше мысал:

```
>>> from numpy import *
>>> A = array([[4,-2,1],[-2,4,-2],[1,-2,3]],float)
>>> b = array([1,4,3],float)
>>> print(diagonal(A)) # Бас диагональ
[ 4. 4. 3.]
>>> print(diagonal(A,1)) # Бірінші қосалғы диагональ
[-2. -2.]
>>> print(trace(A)) # Диагональ элементтерінің қосындысы
11.0
>>> print(argmax(b)) # Ең үлкен элементтің индексі
1
>>> print(argmin(A,axis=0)) # Бағандағы ең кіші элементтердің индексі
[1 0 1]
>>> print(identity(3)) # Бірлік матрица
[[ 1. 0. 0.]
[ 0. 1. 0.]
[ 0. 0. 1.]]
```

Сызықтық алгебра модулі

`numpy` модулі матрицаның керісін табу және сызықты теңдеулерді шешу сияқты бір-бірімен байланысқан есептерді қамтитын `linalg` деп аталатын сызықтық алгебра модульмен бірге келеді. Мысалы,

```
>>> from numpy import array
>>> from numpy.linalg import inv,solve
>>> A = array([[ 4.0, -2.0, 1.0], \
[-2.0, 4.0, -2.0], \
[ 1.0, -2.0, 3.0]])
>>> b = array([1.0, 4.0, 2.0])
>>> print(inv(A)) # Кері матрица
[[ 0.33333333 0.16666667 0. ]
[ 0.16666667 0.45833333 0.25 ]
[ 0. 0.25 0.5 ]]
>>> print(solve(A,b)) # [A]{x} = {b} теңдеуін шешеді
[ 1. , 2.5, 2. ]
```

Массивтерді көшіру

Біз алдында айтқандай, егер a өзгеретін нысан болса, мысалы, тізім болса, b = a тағайындау мәлімдемесі жаңа b нысанын тудырмайды, тек терең көшірме деп аталатын a сілтемесін жасайды. Бұл массивтерге де қатысты. a массивінің тәуелсіз көшірмесін жасау үшін numpy модулінде көшіру әдісін пайдаланыңыз:

```
b = a.copy()
```

Векторизациялау алгоритмдері

Кейде numpy модуліндегі математикалық функциялардың кең қасиеттері кодтағы циклдарды ауыстыру үшін пайдаланылуы мүмкін. Бұл процедура векторизация деп аталады. Мысалы, өрнекті қарастырайық

$$s = \sum_{i=0}^{100} \sqrt{\frac{i\pi}{100}} \sin\frac{i\pi}{100}$$

Тікелей тәсіл – бұл циклдегі қосындыны есептеуге алып келеді, нәтижесінде келесі «скаляр» код:

```
from math import sqrt,sin,pi
x = 0.0; s = 0.0
for i in range(101):
    s = s + sqrt(x)*sin(x)
    x = x + 0.01*pi
print(s)
```

Алгоритмнің векторизацияланған нұсқасы

```
from numpy import sqrt,sin,arange
from math import pi
x = arange(0.0, 1.001*pi, 0.01*pi)
print(sum(sqrt(x)*sin(x)))
```

Бірінші алгоритм `math` модульде `sqrt` және `sin` функцияларының скалярлық нұсқаларын пайдаланатынын, ал екінші алгоритм бұл функцияларды `numpy` ішінен импорттайтынын ескеріңіз. Векторланған алгоритм әлдеқайда жылдамырақ орындалады, бірақ көбірек жадты пайдаланады.

`matplotlib.pyplot` арқылы график құру

`matplotlib.pyplot` модулі Python тілін MATLAB стиліндегі функционалдылықты қамтамасыз ететін 2D сызу функцияларының жиынтығы. Модуль негізгі Python бөлігі емес, оны бөлек орнатуды қажет етеді. Синус және косинус функцияларын тұрғызатын келесі бағдарлама модульді қарапайым ху графиктеріне қолдануды суреттейді.

```
import matplotlib.pyplot as plt
from numpy import arange,sin,cos
x = arange(0.0,6.2,0.2)
plt.plot(x,sin(x),'o-',x,cos(x),'s-') # Белгіленген сызық мәнері пен
                                      # маркері бар сызба
plt.xlabel('x') # x осіне белгі қосу
plt.legend(('синус','косинус'),loc = 0) # loc.3 легенданы қосу
plt.grid(True) # координаталық торды қосу
plt.savefig('testplot.png',format='png') # Суретті png форматта сақтау
plt.show() # суретті экранда көрсету
```

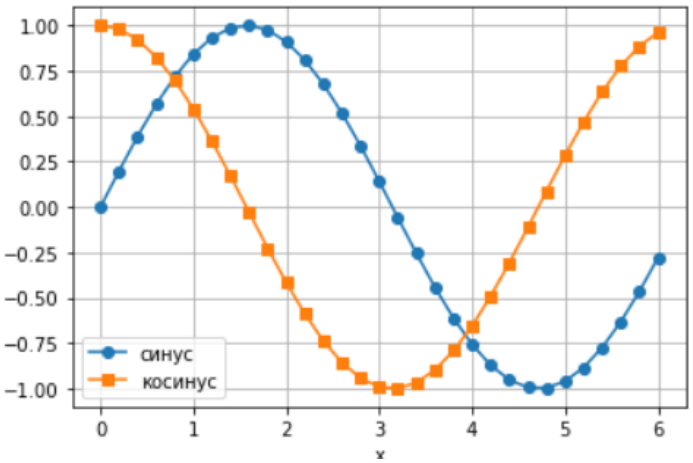
Жол және маркер мәнерлері келесі кестеде көрсетілген жол таңбалары арқылы көрсетіледі (қол жетімді таңбалардың кейбірі ғана көрсетілген).

'-'	Тұтас сызық
'--'	Үзік сызық
'-.'	Сызық-нүкте сызығы
':'	нүктелі сызық
'o'	Шеңбер маркері
's'	Шаршы маркері
'h'	Алтыбұрышты маркер
'x'	x маркері

Легенданы орналастыруға арналған кейбір орын (loc) кодтары

0	"Ең жақсы" орын
1	Жоғарғы оң жақ
2	Жоғарғы сол жақ
3	Төменгі сол жақ
4	Төменгі оң жақ

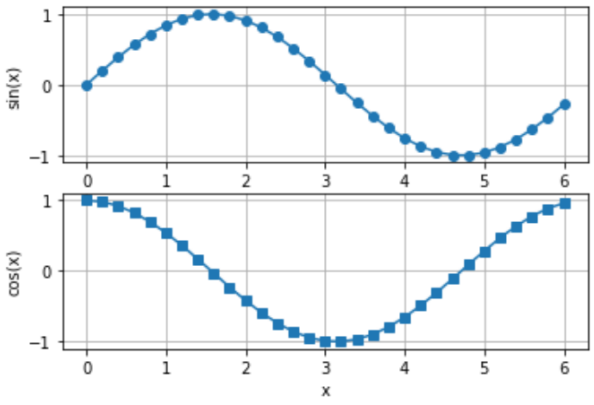
Бағдарламаны іске қосу келесі экранды шығарады:



Суретте бірден көп график болуы мүмкін, бұл келесі кодпен көрсетілген:

```
import matplotlib.pyplot as plt
from numpy import arange,sin,cos
x = arange(0.0,6.2,0.2)
plt.subplot(2,1,1)
plt.plot(x,sin(x),'o-')
plt.xlabel('x');plt.ylabel('sin(x)')
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(x,cos(x),'s-')
plt.xlabel('x');plt.ylabel('cos(x)')
plt.grid(True)
plt.show()
```

Subplot (жолдар, бағандар, сызба нөмірі) командасы ағымдағы суретте ішкі график терезесін орнатады. жол және баған параметрлері суретті жол×баған ішкі сызбалардың торына бөледі (бұл жағдайда екі жол және бір баған). Параметрлер арасындағы үтірлерді алып тастауға болады. Жоғарыдағы бағдарламаның нәтижесі



Атаулар кеңістігі — айнымалылардың атаулары мен олардың мәндерін қамтитын сөздік. Атау кеңістігі бағдарлама іске қосылғанда автоматты түрде жасалады және жаңартылады. Python тілінде атаулар кеңістігінің үш деңгейі бар:

1. Жергілікті атаулар кеңістігі функция шақырылған кезде жасалады. Ол функцияға аргументтер ретінде берілген айнымалыларды және функция ішінде жасалған айнымалыларды қамтиды. Функция аяқталған кезде атаулар кеңістігі жойылады. Егер айнымалы функция ішінде жасалса, оның ауқымы функцияның жергілікті атаулар кеңістігі болып табылады. Ол функцияның сыртында көрінбейді.

2. Жаһандық атаулар кеңістігі модуль жүктелген кезде жасалады. Әрбір модульдің өз атаулар кеңістігі бар. Жаһандық атаулар кеңістігінде тағайындалған айнымалылар модуль ішіндегі кез келген функцияға көрінеді.

3. Интерпретатор іске қосылғанда орнатылған атаулар кеңістігі жасалады. Онда Python интерпретаторымен бірге келетін функциялар бар. Бұл функцияларға кез келген бағдарлама бөлігі арқылы қол жеткізуге болады.

Функцияны орындау кезінде атау кездескен кезде, интерпретатор төменде көрсетілген ретпен іздеу арқылы оны шешуге тырысады: (1) жергілікті атаулар кеңістігі, (2) жаһандық атаулар кеңістігі және (3) орнатылған атаулар кеңістігі. Егер атауды шешу мүмкін болмаса, Python NameError ерекше жағдайын тудырады.

Жаһандық атаулар кеңістігінде орналасқан айнымалылар модуль ішіндегі функцияларға көрінетіндіктен, оларды функцияларға аргументтер ретінде беру қажет емес (бірақ мұны істеу жақсы бағдарламалау тәжірибесі), келесі бағдарлама суреттейді:

```
def divide():
    c = a/b
    print('a/b =',c)
a = 100.0
b = 5.0
divide()
```

```
a/b = 20.0
```

c айнымалысы `divide` функция ішінде жасалғанын және сондықтан функциядан тыс мәлімдемелер үшін қол жетімді емес екенін ескеріңіз. Демек, басып шығару мәлімдемесін функциядан шығару әрекеті сәтсіз аяқталады:

```
def divide():
    c = a/b
a = 100.0
b = 5.0
divide()
print('a/b =',c)
```

```
NameError                                 Traceback (most recent call last)
<ipython-input-2-8a1f073cadd6> in <module>
      4 b = 5.0
      5 divide()
----> 6 print('a/b =',c)

NameError: name 'c' is not defined
```