Məlumatverici mətnlər tərcümə edilmişdir. Kod şərhləri orjinal dildə saxlanılmışdır. Paylaşarkən mənbələrə istinad etməyi unutmayın.
Linkedin: Ramazan Nuhbalayev

Python defolt data tipləri bunlardır

Pythonda istifadə edilən data tipləri

Data tipi	İstifadə məqsədi nədir?	Nümunə
integer	Tam ədədləri ifadə edər	1, 2, 3, 0, -1, -2, -3 gibi
string	Mətn və yazı verilənlərini ifadə edər Mətn verilənləri dırnaq içərisində verilər	"Azərbaycan" və ya 'Azərbaycan' kimi
float	Həqiqi (Real) ədədləri ifadə edər	1.25, 0.75 gibi
boolean	Doğru və ya Yanlışı ifadə edir	Yazılma_statusu = True kimi
complex	Kompleks(Xəyali) ədələri ifadə edər.	1 + 2j, 1.3 + 7.5j kimi

NumPy Data Tipləri

NumPy, Python'dan fərqli olaraq bir çox ədədi data tiplərini dəstəkləyir. Həm də, NumPy-da özünəməxsus bəzi data tipləri də mövcuddur. Misal tamədədlər üçün *i*, müsbət tam ədədlər üçün *u* kimi bir simvol ilə ifadə edilən data tipləri istifadə edilə bilər. Aşağıda NumPy'dakı data tipləri ve bunları ifadə edən simvolları göstərən bir siyahı verərik.

Numpy Data Tipləri

Numpy data tipi	Açıqlama
i	Tam ədəd
b	Bool dəyişəni
u	Müsbət tam ədəd
f	Onluq kəsr
c	Kompleks(Xəyali) ədədlər
m	timedelta
M	datetime
O	Object
S	mətn
U	Unikod string
V	digər növlər üçün sabit yaddaş yığını

NumPy, C dilindəki kimi ibtidai data tiplerini dəstəkləyir. Aşağıdaki siyahıya baxaq

Platformadan asılı *olaraq* dəyişən Numpy data tipləri

Numpy data tipi	C-dəki forması	Açıqlama
numpy.bool_	bool	Boolean (True or False) stored as a byte
numpy.byte	signed char	Platformadan asılıdır
numpy.ubyte	unsigned char	Platformadan asılıdır
numpy.short	short	Platformadan asılıdır
numpy.ushort	unsigned short	Platformadan asılıdır
numpy.intc	int	Platformadan asılıdır
numpy.uintc	unsigned int	Platformadan asılıdır
numpy.int_	long	Platformadan asılıdır
numpy.uint	unsigned long	Platformadan asılıdır
numpy.longlong	long long	Platformadan asılıdır
numpy.ulonglong	unsigned long long	Platformadan asılıdır
numpy.half numpy.float16		Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
numpy.single	float	Platform-defined single precision float: typically sign bit, 8 bits exponent, 23 bits mantissa
numpy.double	double	Platform-defined double precision float: typically sign bit, 11 bits exponent, 52 bits mantissa.
numpy.longdouble	long double	Platform-defined extended-precision float
numpy.csingle	float complex	Complex number, represented by two single-precision floats (real and imaginary components)
numpy.cdouble	double complex	Complex number, represented by two double-precision floats (real and imaginary components).

Platformadan asılı *olaraq* dəyişən Numpy data tipləri

Numpy data tipi	C-dəki forması	Açıqlama
numpy.clongdouble	long double complex	Complex number, represented by two extended-precision floats (real and imaginary components).

Platformadan asılı *olmayaraq* dəyişən Numpy data tipləri

Numpy data tipi	C-dəki forması	Açıqlama
np.int8	int8_t	Byte (-128 ile 127 arasi)
np.int16	int16_t	Tam ədəd (-32768 ilə 32767 arası)
np.int32	int32_t	Tam ədəd (-2147483648 ilə 2147483647 arası)
np.int64	int64_t	Tam ədəd (-9223372036854775808 ilə 9223372036854775807 arası)
np.uint8	uint8_t	Müsbət tam ədəd (0 ilə 255 arası)
np.uint16	uint16_t	Müsbət tam ədəd (0 ilə 65535 arası)
np.uint32	uint32_t	Müsbət tam ədəd (0 ilə 4294967295 arası)
np.uint64	uint64_t	Müsbət tam ədəd (0 ilə 18446744073709551615 arası)
np.intp	intptr_t	İndeksləmək üçün istifadə edilən Tam ədəd
np.uintp	uintptr_t	Bir pointeri saxlaya biləcək böyük tam ədəd
np.float32	float	
np.float64 np.float_	double	Pythondakı mövcud float ilə eyni dəqiqliyə malikdir
np.complex64	float complex	Kompleks ədəd, 2 dənə 32-bit float dəyər ilə göstərilir (həqiqi hissə + xəyali hissə)
np.complex128 np.complex_	double complex	Pythondakı mövcud kompleks data tipi ilə ilə eyni dəqiqliyə malikdir

Adlarının yanında ədədlər olan data tipləri, data tipinin bit ölçüsünü göstərir (yani, yaddaşda tək bir dəyişəni ifadə etmək üçün lazım olan yaddaş tutumunu). İnt və intp kimi bəzi tiplər, platformalardan asılı olaraq fərqli bit ölçüsünə malikdir. (misal 32 bit və 64 bit cihazlardan asılı olaraq). Aşağı səviyyə kodlarda (C, Fortran kimi dillərdə yazılmış) interfeys yaradarkən bunlara diqqət yetirmək lazımdır.

```
import numpy as np
x = np.float32(155)
y = np.int_([10,112,40000])

print(x)
print(y)
```

```
155.0
[   10   112 40000]
```

NumPy array obyektinin, array -ın hansı data tipində olduğunu qaytaran *dtype* adlı bir xüsusuiyyəti var.
dtype, dəyişənin data tipi haqqında bit ölçüsü ve bayt sırası kimi məlumatları saxlayır.

```
import numpy as np
x=np.array([1,2,3])

print(x.dtype) #x in data tipi nədir

y=np.int8(x) #x in data tipini dəyişək

print(y.dtype) #yeni dəyər y in data tipi
```

```
int32
int8
```

Mövcud bir massivin data tipini dəyişmənin ən asan yolu, massivin astype metodu ilə bir kopyasını yaradarıq. Ardınca istədiyimiz data tipini funksiyanın parametri olaraq təyin edərik.

Data tipi, float üçün ‘f’, tam ədəd üçün ‘i’ formasında bir string istifadə edilərək ifadə edilə bilər və ya data tipini birbaşa float və ya tam ədəd kimi daxil edə bilərsiniz.

```
import numpy as np

dizi1 = np.array([2.4, 3, 1.6, 0]) #list dizi nesnesini numpy dizisine dönüştürelim
print(dizi1) #çıktı-satır 1. numpy dizisini yazdıralım
```

```
print(dizi1.dtype) #çıktı-satır 2. numpy dizisinin veri tipi nedir?

#bir önceki dizinin veri tipini integer yapalım
#aşağıdaki komut satırında i yerine int de yazılabilir
dizi1_new = dizi1.astype('i')
print(dizi1_new) #çıktı satır 3. yeni diziyi yazdıralım
print(dizi1_new.dtype) #çıktı satır 4. yeni dizinin veri tipini sorgulayalım.

#list, tuple ve çeşitli veri tiplerinden oluşan bir dizi tanımlayalım
dizi3 = np.array([[1,2.0],[0,0],(1+1j,3.)])
print(dizi3.dtype) #çıktı satır 5. bu dizinin veri tipi nedir?
print(dizi3) #çıktı satır 6. bu diziyi yazdıralım
```

```
[2.4 3.  1.6 0. ]
float64
[2 3 1 0]
int32
complex128
[[1.+0.j 2.+0.j]
 [0.+0.j 0.+0.j]
 [1.+1.j 3.+0.j]]
```

Python-da Massivə Bənzər Quruluşların NumPy-də Massivlərə çevrilməsi:

Python-da massivə bənzər strukturda düzülmüş ədədi verilənlər array() funksiyasının köməyi ilə NumPy massivlərinə çevrilə bilər. Python-da ən bariz massiv -ə bənzəyən strukturlar siyahılar və tuplelərdir. Python-da digər massiv -ə bənzər strukturların array() funksiyasından istifadə edərək NumPy massivinə çevrilə biləcəyini bilmək istəyirsinizsə, cəhd edin və onun işləyib-işləmədiyini yoxlayın! NumPy-də massivlər yaratmaq üçün array() funksiyasından istifadə edirik, bu barədə əvvəlki bölmədə danışmışdıq. Bu funksiyanın gözəl xüsusiyyəti ixtiyari olaraq istifadə oluna bilər. Array() funksiyası dtype-ni arqument (sərbəst dəyişən) kimi əhatə edə bilər. Beləliklə, massiv elementlərini hansı məlumat növünə görə təyin edəcəyinizi əvvəldən qərar verə bilərsiniz. Aşağıdakı nümunədə ədədi NumPy massivinin hər bir elementi 5 bayt olan string massivinə çevrildi.

```
import numpy as np

#veri tipi string olan bir numpy dizisi oluşturalım
dizi0 = np.array([12, 200, 3500, 40000, 5], dtype='S')

print(dizi0)
print(dizi0.dtype)
```

```
[b'12' b'200' b'3500' b'40000' b'numpy']
|S5
```

NumPy-də istifadə olunan i, u, f, S və U məlumat tiplərində ölçüsünü əvvəlcədən müəyyən etmək mümkündür. Məsələn, mövcud ədədi dəyərləri saxlayan massivdən 4 baytlıq tam ədədlərdən ibarət massiv yaradaq.

```
import numpy as np

#veri tipi 4 byte integer olan bir numpy dizisi oluşturalım
dizi0 = np.array([12, 200, 3500, 40000, 5], dtype='i4')

print(dizi0)
print(dizi0.dtype)
```

```
[   12   200  3500 40000     5]
int32
```

Bunu bilmək yaxşıdır!

Əgər massiv elementləri həm sətir, həm də ədədi dəyərlərdən ibarətdirsə, massiv string data tipinə çevrilə bilər, lakin ədədi data tipinə çevrilməyə cəhd edilərsə, NumPy ValueError xətası verəcək. ValueError xətası funksiyaya ötürülən arqument növü gözlənilməyən və ya səhv olduqda baş verir.

```
import numpy as np

#dizi elemanları hem string hem sayısal değerler alıyorsa
#String veri tipine dönüştürülebilir
#ancak sayısal veri tiplerine dönüştürülemez.
dizi0 = np.array([12, 200, 35, 4, 'numpy'], dtype='S')

print(dizi0) #çıktı satır 1- dizi elemanlarını yazdıralım
```

```
print(dizi0.dtype) #çıktı satır 2- dizi elemanlarının veri tipi nedir?
```

```
[b'12' b'200' b'35' b'4' b'numpy']
|S5
```

```
import numpy as np
#'numpy' gibi bir string değer tamsayıya dönüştürülemez(hata oluşur)
dizi1 = np.array([12, 200, 35, 4, 'numpy'], dtype='int')

print(dizi1.dtype) # dizi elemanlarının veri tipi eşleşmiyor: ValueError
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-21-35376652d31d> in <module>
      1 import numpy as np
      2 #'numpy' gibi bir string değer tamsayıya dönüştürülemez(hata oluşur)
----> 3 dizi1 = np.array([12, 200, 35, 4, 'numpy'], dtype='int')
      4
      5 print(dizi1.dtype) # dizi elemanlarının veri tipi eşleşmiyor: ValueError

ValueError: invalid literal for int() with base 10: 'numpy'
```

SEARCH STACK OVERFLOW

NumPy-də sıfırdan massiv yaratmaq NumPy sıfırdan ibarət massivlər yaratmaq üçün built-in funksiyalara malikdir. Bu funksiyalardan biri **zeros(shape)** göstərilən formada 0 dəyəri ilə doldurulmuş massiv yaradır. Onun standart dtipi float64-dür. Aşağıdakı nümunəni nəzərdən keçirək.

```
#NumPy'da sıfırdan array oluşturmak
import numpy as np

#zeros belirtilen şekle sahip 0 değerlerle dolu bir dizi oluşturur
dizi= np.zeros((2, 3))

print(dizi)
print(dizi.dtype)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
float64
```

Eynilə, ones(shape) göstərilən formada 1 dəyəri ilə dolu massiv yaradar. Bütün digər xüsusiyyətlər zeros() ilə eynidir.

```
import numpy as np

#ones belirtilen şekle sahip 1 değerleriyle dolu bir dizi oluşturur
dizi= np.ones((2, 3))

print(dizi)
print(dizi.dtype)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
float64
```

arange() funksiyası müntəzəm artan dəyərlərdən ibarət massivlər yaradır. Aşağıdakı nümunəni nəzərdən keçirək.

```
import numpy as np
#arange düzenli olarak artan değerlere sahip diziler oluşturur.
dizi1=np.arange(10) #başlangıç belirtilmezse 0'dan başlar

dizi2=np.arange(3, 12, dtype=float) #son değeri almadığına dikkat!

dizi3=np.arange(3, 5, 0.7) #3'ten 5'e kadar 0.7 artarak

print(dizi1) #çıktı-satır 1
print(dizi2) #çıktı-satır 2
print(dizi3) #çıktı-satır 3
```

```
[0 1 2 3 4 5 6 7 8 9]
[ 3.  4.  5.  6.  7.  8.  9. 10. 11.]
[3.  3.7 4.4]
```

Bunu bilmək yaxşıdır!

Arange() funksiyasına bənzər şəkildə istifadə edilən linspace() verilmiş sayda elementlərlə və müəyyən edilmiş başlanğıc və son qiymətlər arasında bərabər məsafəli massivlər yaradır. linspace() funksiyasının üstünlüyü odur ki, arange() adətən ixtiyari başlanğıc, dayanma və addım dəyərləri yaradır, lakin linspace() elementlərin sayına, başlanğıc və son nöqtəyəni biz təyin edərək müəyyənləşdiririk.

```
import numpy as np

z = np.linspace(3., 5., 5)
# artım değerinin 5 girildiğine, çıktıda 0.5 olarak
# algılandığına dikkat edin!
print(z)
```

```
[3.  3.5 4.  4.5 5. ]
```

Ədədi data tiplərinin sabit ölçüsü mövcud olandan daha çox yaddaş tələb etdikdə NumPy daşma xətalarına səbəb ola bilər. Məsələn, numpy.power() funksiyası(qüvvətə yüksəldmə funksiyası)nın aşağıdakı nümunəsini nəzərdən keçirək .

```
import numpy as np

x = np.power(100, 8, dtype=np.int64)

y = np.power(100, 8, dtype=np.int32) #overflow error

print(x)
print(y)
```

```
10000000000000000
1874919424
```

Yuxarıdakı misalda NumPy 100 üstü 8 -i qədər 64 bitlik tam ədədlər üçün doğru hesablayır, lakin 32 bitlik tam ədədlər üçün 1874919424 kimi yanlış dəyər yaradır.

Tam ədəd data tipinin davranışı NumPy və Python-da eyni deyil. Başqa sözlə, daşma xətaları baxımından Python -da istifadə olunan int və NumPy -da istifadə olunan bir-birindən fərqlənir. Python-da int ölçüsü çevikdir. Yəni Python int data tipi istənilən tam ədədi ala biləcək qədər genişlənə bilir.

Bunu bilmək yaxşıdır!

NumPy istifadə edərkən istifadə etdiyiniz platformadan asılı olaraq, numpy.iinfo(data_tipi) funksiyası int və float məlumat növlərinin ala biləcəyi minimum və ya maksimum dəyərləri tapmaq üçün istifadə edilə bilər. Aşağıdakı nümunəni nəzərdən keçirək.

```
import numpy as np
print(np.iinfo(int)) #kullandığım sistemdeki varsayılan int değerleri nedir?
print(np.iinfo(np.int32)) #32-bit integer sınırları nelerdir?
print(np.iinfo(np.int64)) #64-bit integer sınırları nelerdir?
```

```
Machine parameters for int32
---------------------------------------------------------------
min = -2147483648
max = 2147483647
---------------------------------------------------------------

Machine parameters for int32
---------------------------------------------------------------
min = -2147483648
max = 2147483647
---------------------------------------------------------------

Machine parameters for int64
---------------------------------------------------------------
min = -9223372036854775808
max = 9223372036854775807
---------------------------------------------------------------
```

Əgər 64 bitlik tam ədədlər sizə hələ də kiçik gəlirsə və istədiyiniz nəticəni ala bilmirsinizsə, nəticə float-a çevrilə bilər. Float ədədləri daha genişdir ədədlər çoxluğunu əhatə edir, lakin sizə qeyri-müəyyən təxmini dəyərlər verir.

```
import numpy as np

print(np.power(100, 100, dtype=np.int64)) #çıktı için 64 bit integer yeterli değil-overflow
print(np.power(100, 100, dtype=np.float64)) #float' a çevirelim
```

```
0
1e+200
```

NumPy-də massivlər yaratmaq üçün müxtəlif üsullardan istifadə etmək olar:
NumPy-də massiv obyektlərini (məsələn, arange(), ones(), zeros()) Python-da massiv kimi strukturlardan (list, tuple kimi) çevirməklə əldə etmək olar Bu bölmədə bu iki metodun necə istifadə edildiyini təsvir etdim. Bununla belə, bu üsullardan başqa NumPy massivləri standart series -lərdən və ya digər xüsusi formatlardan oxumaqla, eləcə də kitabxanaların hazır funksiyalarından (məsələn, random(), indices()) istifadə etməklə də yaradıla bilər.

Kaynaklar: https://numpy.org/, https://www.w3schools.com/ , NumPy Veri Tipleri ve NumPy'da Array Oluşturma – Bölüm 7

Bu bölümün Github resursları üçün açın.