Working With Lists

Looping Through an Entire List

Çox vaxt bir list-də olan bütün elementləri ardıcıllıqla keçmək və hər element üzərində eyni tapşırığı yerinə yetirmək istəyəcəksən. Məsələn, bir oyunda ekrandakı hər elementi eyni qədər hərəkət etdirmək lazım ola bilər, və ya bir list dolusu rəqəmlərin hər birinə eyni statistik əməliyyatı tətbiq etmək istəyə bilərsən. Və ya bəlkə də bir website-dəki məqalələrin başlıqlarını biri–birinin ardınca göstərmək istəyəcəksən.
Bir list-dəki **hər element üçün eyni hərəkəti** yerinə yetirmək lazım olduqda, Python-da **for loop** istifadə olunur.

Tutaq ki, magicians adında bir list var və biz bu list-dəki hər bir adı ekrana çıxarmaq istəyirik. Bunu hər adı ayrı-ayrılıqda list-dən götürüb print etməklə edə bilərik, amma bu yanaşma bir neçə problem yaradar.
Birincisi, list uzun olanda bu üsul çox təkrarlayıcı və yorucu olar.
İkincisi, list-in uzunluğu dəyişəndə kodu hər dəfə yenidən düzəltməli olardıq.
For loop isə bu iki problemi aradan qaldırır, çünki bütün prosesi Python-un özünə həvalə edir.

Gəlin bir for loop istifadə edərək list-dəki hər bir adı çapa verək:

magicians.py

```
magicians = ['alice', 'david', 'carolina']

for magician in magicians:
    print(magician)
```

əvvəlcə magicians adlı list-i müəyyən edirik. for loop yazırıq. Bu sətr Python-a deyir ki, magicians list-indən bir adı götür və onu magician adlı dəyişənə təyin et. Python-a deyirik ki, hal-hazırda magician dəyişəninə verilmiş adı print etsin.

A Closer Look at Looping

Looping anlayışı vacibdir, çünki kompüterlərin təkrarlanan tapşırıqları avtomatlaşdırmaq üçün ən çox istifadə etdiyi üsullardan biridir. Məsələn, magicians.py faylında istifadə etdiyimiz sadə bir loop-da Python əvvəlcə loop-un ilk sətrini oxuyur:

```
for magician in magicians:
```

Bu sətir Python-a deyir ki, magicians adlı list-dən ilk dəyəri götür və onu magician adlı variable-a təyin et. Bu ilk dəyər `'alice'` olur. Sonra Python növbəti sətri oxuyur:

```
print(magician)
```

Python hal-hazırda magician variable-ının dəyərini—hələ də `'alice'` olan dəyəri—çap edir. List-də başqa dəyərlər olduğu üçün Python yenidən loop-un ilk sətrinə qayıdır:

```
for magician in magicians:
```

Bu dəfə Python list-dəki növbəti adı—`'david'`—götürür və həmin dəyəri magician variable-ına təyin edir. Sonra yenə aşağıdakı sətri icra edir:

```
print(magician)
```

Python bu dəfə magician-ın yeni dəyərini çap edir—`'david'`. Python loop-u bir dəfə də **list-dəki son dəyər** olan `'carolina'` üçün təkrar edir. List-də daha dəyər qalmadıqda Python proqramın növbəti sətrinə keçir. Bu nümunədə for loop-dan sonra heç nə olmadığı üçün proqram bitir.

Loop-larla ilk dəfə işləyəndə yadda saxla ki, **list-də neçə element varsa**, həmin addımlar **bir o qədər dəfə** təkrar olunacaq. Əgər list-də 1 milyon element varsa, Python həmin addımları 1 milyon dəfə icra edəcək—və bunu adətən çox sürətli edir.

Həmçinin, öz for loop-larını yazarkən unutma ki, list-dəki hər dəyərə təyin olunan müvəqqəti variable-ın adını sən istədiyin kimi seçə bilərsən. Lakin list-in məzmununa uyğun **mənalı bir ad** seçmək daha yaxşıdır. Məsələn, bu for loop-lar düzgün ad seçiminə nümunədir:

```
for cat in cats:
for dog in dogs:
for item in list_of_items:
```

Doing More Work Within a for Loop

For loop içində hər elementlə istədiyin qədər iş görə bilərsən. Gəlin əvvəlki nümunəni genişləndirərək hər bir magician üçün xüsusi bir mesaj çap edək:

magicians.py

```
magicians = ['alice', 'david', 'carolina']

for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
```

Output:

```
Alice, that was a great trick!
David, that was a great trick!
Carolina, that was a great trick!
```

For loop daxilində istədiyin qədər sətir yaza bilərsən

For loop-un başlayış sətrindən sonra indent edilmiş bütün sətirlər loop-un içində hesab olunur və list-dəki **hər element üçün bir dəfə** icra edilir.

```
magicians = ['alice', 'david', 'carolina']

for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
    print(f"I can't wait to see your next trick, {magician.title()}.\n")
```

Burada hər iki print() sətri indent edildiyi üçün hər magician üçün **iki mesaj** çap olunacaq.
İkinci print-dəki `"\n"` isə hər dövrədən sonra boş sətir əlavə edir ki, mesajlar səliqəli görünsün.

Output:

```
Alice, that was a great trick!
I can't wait to see your next trick, Alice.

David, that was a great trick!
I can't wait to see your next trick, David.

Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.
```

Doing Something After a for Loop

Bəs for loop bitəndən sonra nə baş verir? Adətən, bütün loop əməliyyatları bitəndən sonra nəticəni yekunlaşdırmaq və ya proqramın növbəti tapşırıqlarına keçmək lazım olur.

For loop-dan **sonra indent edilməmiş** sətirlər **yalnız bir dəfə icra olunur**.

Məsələn, indi bütün magician-lara ümumi təşəkkür mesajı verək. Bu mesaj for loop-dan sonra, **indent edilməmiş şəkildə** yazılır:

```
magicians = ['alice', 'david', 'carolina']

for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
    print(f"I can't wait to see your next trick, {magician.title()}.\n")
```

```
print("Thank you, everyone. That was a great magic show!")
```

Output:

```
Alice, that was a great trick!
I can't wait to see your next trick, Alice.

David, that was a great trick!
I can't wait to see your next trick, David.

Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.

Thank you, everyone. That was a great magic show!
```

Making Numerical Lists

Rəqəmləri saxlamaq üçün bir çox səbəb var. Məsələn, bir oyunda hər xarakterin mövqeyini izləmək, oyunçunun yüksək ballarını saxlamaq istəyə bilərsiniz. Məlumat vizuallaşdırmalarında isə tez-tez temperatur, məsafə, əhali sayı və ya coğrafi koordinatlar kimi rəqəmsal məlumatlarla işləyirsiniz.

Python siyahılar rəqəmləri saxlamaq üçün idealdır və Python siyahıları ilə effektiv işləməyiniz üçün müxtəlif alətlər təqdim edir. Bu alətləri öyrəndikdə, milyonlarla elementdən ibarət siyahılarla da problemsiz işləyə bilərsiniz.

range() funksiyasından istifadə:

Python-un `range()` funksiyası bir sıra rəqəmlər yaratmağı asanlaşdırır. Məsələn:

```
for value in range(1, 5):
    print(value)
```

Bu kod 1-dən 5-ə qədər rəqəmləri çap edəcək kimi görünür, amma əslində 5-i çap etmir. Çünki `range()` funksiyası başlanğıc dəyərdən başlayır və ikinci dəyərə çatanda dayanır. Son dəyər daxil olmur. 1-dən 5-ə qədər çap etmək üçün:

```
for value in range(1, 6):
    print(value)
```

Əgər yalnız bir arqument verilsə, `range()` 0-dan başlayır. Məsələn, `range(6)` 0-dan 5-ə qədər rəqəmləri qaytarır.

range() ilə siyahı yaratmaq

`range()` funksiyasının nəticəsini birbaşa `list()` ilə siyahıya çevirə bilərsiniz:

```
numbers = list(range(1, 6))
print(numbers)
```

Output:

[1, 2, 3, 4, 5]

`range()`-in üçüncü arqumenti ilə addım ölçüsünü də təyin etmək olar. Məsələn, 1-dən 10-a qədər cüt rəqəmlər:

```
even_numbers = list(range(2, 11, 2))
print(even_numbers)
```

Output:

[2, 4, 6, 8, 10]

Kvadrat rəqəmlərin siyahısını yaratmaq

Birinci 10 tam ədədin kvadratını siyahıya əlavə etmək üçün:

```
squares = []
for value in range(1, 11):
    square = value ** 2
    squares.append(square)
print(squares)
```

Output:

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

Daha qisa yazmaq:

```
squares = []
for value in range(1, 11):
    squares.append(value ** 2)
print(squares)
```

Siyahı ilə sadə statistik əməliyyatlar

Python siyahısı üzərində aşağıdakı funksiyalardan istifadə edə bilərsiniz:

```
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
min(digits)  # 0
max(digits)  # 9
sum(digits)  # 45
```

List Comprehensions (Siyahı Qısa Yazımı)

List comprehension ilə eyni siyahını bir sətirdə də yarada bilərsiniz:

```
squares = [value**2 for value in range(1, 11)]
print(squares)
```

List comprehension həm döngü, həm də yeni element yaratma əməliyyatını birləşdirir və hər yeni elementi avtomatik siyahıya əlavə edir. Bu üsul çox qısa və praktikdir.

Working with Part of a List

Siyahını dilimləmək (Slicing a List)

Python-da siyahının bütün elementlərini işləməyi öyrəndikdən sonra, **siyahının yalnız müəyyən bir hissəsini** işləmək də mümkündür. Python bunu **slice (dilim)** adlandırır.

- Dilim yaratmaq üçün **başlanğıc və son indeksləri** göstərirsiniz.
- Python **son indeksi daxil etmir**, yəni sonuncudan bir əvvəl dayanar.

Məsələn, bir komandadakı oyunçular siyahısının ilk üç elementini çap etmək:

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[0:3])
```

Output:
['charles', 'martina', 'michael']
Əgər sonuncu indeksi qeyd etməsəniz, Python avtomatik olaraq **siyahının sonuna qədər** götürür.

Əgər başlanğıc indeksi qeyd etməsəniz, Python avtomatik olaraq **siyahının əvvəlinə** başlayır.

Negativ indekslər isə **siyahının sonundan** saymağa imkan verir:

```
print(players[-3:])  # ['michael', 'florence', 'eli']
```

Dilimlə `for` döngüsündə işləmək

Dilimlərdən `for` döngüsündə istifadə edə bilərsiniz:

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print("Here are the first three players on my team:")
for player in players[:3]:
    print(player.title())
```

Output:
Here are the first three players on my team:
Charles
Martina

Michael
Burada Python yalnız ilk üç oyunçu ilə işləyir, bütün siyahı ilə deyil.
Siyahını kopyalamaq (Copying a List)
Bəzən mövcud siyahıdan yeni, tamamilə müstəqil bir siyahı yaratmaq istəyirik. Bunun üçün **[:]** istifadə edirik.

```
my_foods = ['pizza', 'falafel', 'carrot cake']
friend_foods = my_foods[:]  # my_foods siyahısının kopyası
print("My favorite foods are:")
print(my_foods)
print("\nMy friend's favorite foods are:")
print(friend_foods)
```

- Hər iki siyahı eyni elementləri ehtiva edir, amma **müstəqildir**:

```
my_foods.append('cannoli')

friend_foods.append('ice cream')

print(my_foods)      # ['pizza', 'falafel', 'carrot cake', 'cannoli']

print(friend_foods)  # ['pizza', 'falafel', 'carrot cake', 'ice cream']
```

Əgər sadəcə `friend_foods = my_foods` yazsaq, **hər iki dəyişən eyni siyahıya işarə edəcək**.

Belə olduqda hər iki siyahıda əlavə olunan elementlər görünəcək.

```
friend_foods = my_foods  # bu düzgün kopya yaratmır
```

['pizza', 'falafel', 'carrot cake', 'cannoli', 'ice cream']

Ona görə, siyahını müstəqil kopyalamaq üçün **[:] dilimindən istifadə etmək vacibdir.**

Tuples

- **Listlər** proqram boyunca dəyişə bilən elementləri saxlamaq üçün yaxşıdır. Məsələn, oyunlardakı xarakterlər və ya vebsaytdakı istifadəçilərin siyahısı.

- Bəzən isə elementləri **dəyişməyən siyahı** yaratmaq lazım olur. Bunun üçün **tuple** istifadə olunur.

- Python-da dəyişməyən elementlər **immutable** adlanır, immutable list isə **tuple** adlanır.

1. Tuple Tərifi

- Tuple, **listə bənzəyir**, amma **kvadrat mötərizə** **[ ]** əvəzinə **dairəvi mötərizə** **( )** istifadə olunur.
- Elementlərə **indeks vasitəsilə** müraciət edə bilərsiniz.

```
dimensions = (200, 50)  # Tuple yaratmaq
print(dimensions[0])    # 200
print(dimensions[1])    # 50
```

Tuple yaratdıqdan sonra **elementləri dəyişdirmək olmaz**:

```
dimensions = (200, 50)
dimensions[0] = 250   # Xəta verəcək
ERROR:
TypeError: 'tuple' object does not support item assignment
```

Yalnız bir elementli tuple yaratmaq üçün **sonuna vergül əlavə etmək lazımdır**:

```
my_t = (3,)
```

Tuple üzərində `for` döngüsü

Tuple-dəki bütün elementlər üzərində **for döngüsü** ilə keçmək olar, listlə eyni şəkildə.

```
dimensions = (200, 50)
for dimension in dimensions:
    print(dimension)
```

Tuple-u Yenidən Təyin Etmək

- Tuple dəyişdirilə bilməsə də, onu təmsil edən **dəyişkənə yeni tuple** təyin etmək olar.

```
dimensions = (200, 50)

print("Original dimensions:")

for dimension in dimensions:

    print(dimension)


dimensions = (400, 100)  # Yeni tuple təyin edirik

print("\nModified dimensions:")

for dimension in dimensions:

    print(dimension)
```

Output:

Original dimensions:

200

50

Modified dimensions:

400

100

Styling Your Code

Kodun oxunaqlı olması proqramınızın nə etdiyini izləməyinizi asanlaşdırır və başqalarının da kodunuzu anlamasına kömək edir.

Python proqramçıları müəyyən stil qaydaları qəbul ediblər ki, bütün kod oxşar struktura malik olsun. Kodunuzu təmiz yazmağı öyrəndikdən sonra, başqalarının Python kodunu da anlaya biləcəksiniz, əgər onlar da eyni qaydalara əməl

edirlərsə. Peşəkar proqramçı olmaq istəyirsinizsə, bu qaydaları mümkün qədər tez öyrənmək yaxşı vərdişdir.

Style Guide

Python-da dil dəyişiklikləri təklif etmək üçün **Python Enhancement Proposal (PEP)** istifadə olunur. Ən məşhurlardan biri **PEP 8**-dir, bu, Python proqramçılarına kodu necə stilizə etmək barədə qaydalar verir. PEP 8 çox genişdir, amma əsasən mürəkkəb kod strukturlarına aid olur.

Kodun oxunması yazmaqdan daha vacibdir. Kod bir dəfə yazılır, sonra isə onu oxumaq, debug etmək və dəyişikliklər etmək üçün çox vaxt sərf olunur. Digər proqramçılarla paylaşdıqda da kod oxunur. Kodun yazılması ilə oxunması arasında seçim edərkən Python proqramçıları **oxunaqlı kod yazmağı** tövsiyə edir.

Əsas Stil Qaydaları

Indentation

- PEP 8 tövsiyə edir ki, **hər indent səviyyəsi üçün 4 boşluq** istifadə olunsun.
- Boşluqlar oxunaqlığı artır və çoxlu indent səviyyələri üçün yer saxlayır.
- Tab düyməsi işlədildikdə, onu boşluqlara çevirmək tövsiyə olunur.
- Tab və boşluqları qarışdırmaq Python-da problemlərə səbəb ola bilər.

Line Length

- Hər sətir **80 simvoldan az** olmalıdır.
- Tarixi səbəblər: köhnə terminal pəncərələri bir sətirə yalnız 79 simvol sığdırırdı.
- Müasir ekranlarda uzun sətirlər işləyir, amma professional proqramçılar bir neçə faylı yan-yana açdıqda oxunması asan olsun deyə bu qaydaya əməl edirlər.
- Şərhlər üçün isə **72 simvol** tövsiyə olunur.

Blank Lines

- Kodun vizual qruplaşdırılması üçün istifadə olunur.

- Məsələn, bir hissə 5 sətrdən ibarətdirsə və digər hissə 3 sətrdən ibarətdirsə, bu iki hissəni **bir boş sətir** ilə ayırmaq kifayətdir.

- 3-4 boş sətir əlavə etmək artıqdır.

- Boş sətirlər kodun işinə təsir etməsə də, oxunaqlılığa təsir edir.