06-for-loop

September 16, 2022

1 For cycle

The for loop is used when looping over a sequence, like a list, tuple, or a collection of objects.

An iterable is an object capable of returning its members one at a time. Examples of iterables include all sequence types (such as list, str, and tuple) and some non-sequence types like dict, or file objects.

Objects of any classes you define with an `__iter__()` or `__getitem__()` method.

1.1 Basic syntax

[1]:
```
for number in [0, 1, 2, 3, 4]:
    print(number)
    print(number ** 2)
```

```
0
0
1
1
2
4
3
9
4
16
```

Iterating over a range

[2]:
```
for number in range(5):
    print(number)
```

```
0
1
2
3
4
```

Iterating over another range [3, 4, 5, 6, 7]

[3]:
```
for number in range(3, 8):
    print(number)
```

```
3
4
5
6
7
```

Iterating even over another range [-10, -6, -2, 2, 6]

[4]:
```
for number in range(-10, 10, 4):
    print(number)
```

```
-10
-6
-2
2
6
```

Iterating over a sequence

[5]:
```
surnames = ['Rivest', 'Shamir', 'Adleman']
for surname in surnames:
    print(surname)
```

```
Rivest
Shamir
Adleman
```

Iterating over a sequence with index - old way

[6]:
```
for position in range(len(surnames)):
    print(position, surnames[position])
```

```
0 Rivest
1 Shamir
2 Adleman
```

Iterating over a sequence with index - Python way

[7]:
```
for position, surname in enumerate(surnames):
    print(position, surname)
```

```
0 Rivest
1 Shamir
2 Adleman
```

Iterating over a sequence with index, position starting at 1

[8]:
```
for position, surname in enumerate(surnames, 1):
    print(position, surname)
```

```
1 Rivest
2 Shamir
3 Adleman
```

In short,

```
- An iterable is an object capable of returning its members one at a time.
- Examples of iterables include all sequence types (such as list, str, and tuple) and some non-
```

1.2 Iterating over multiple sequences

How to iterate over multiple sequences

[9]:
```
people = ['Darwin', 'Bernardo', 'Odysseas', 'Ramos']
ages = [25, 20, 26, 20]

# old way!
for position in range(len(people)):
    person = people[position]
    age = ages[position]
    print(person, age)
```

```
Darwin 25
Bernardo 20
Odysseas 26
Ramos 20
```

more pythonic

[10]:
```
for position, person in enumerate(people):
    age = ages[position]
    print(person, age)
```

```
Darwin 25
Bernardo 20
Odysseas 26
Ramos 20
```

and, the pythonic way

[11]:
```
for person, age in zip(people, ages):
    print(person, age)
```

```
Darwin 25
Bernardo 20
Odysseas 26
Ramos 20
```

[12]:
```
for person_age in zip(people, ages):
    print(person_age)
```

```
('Darwin', 25)
('Bernardo', 20)
('Odysseas', 26)
('Ramos', 20)
```

[13]:
```
nationalities = ['Uruguay', 'Portugal', 'Greek', 'Portugal']

for person, age, nationality in zip(people, ages, nationalities):
    print(person, age, nationality)
```

```
Darwin 25 Uruguay
Bernardo 20 Portugal
Odysseas 26 Greek
Ramos 20 Portugal
```

1.3 continue

The **continue** statement, tells the looping construct (**for** or **while**) to immediately stop execution of the body and go to the next iteration, if any.

[14]:
```
from datetime import date, timedelta
today = date.today()
tomorrow = today + timedelta(days=1) # today + 1 day is tomorrow

products = [
    {'sku': '1', 'expiration_date': today, 'price': 100.0},
    {'sku': '2', 'expiration_date': tomorrow, 'price': 50},
    {'sku': '3', 'expiration_date': today, 'price': 20},
]

for product in products:
    if product['expiration_date'] != today:
        continue                # go to the next product
    product['price'] *= 0.8     # apply 20% discount
    print('Price for sku', product['sku'], 'is now', product['price'])
```

```
Price for sku 1 is now 80.0
Price for sku 3 is now 16.0
```

1.4 break

The **break** statement terminates the current loop and resumes execution at the next statement

[15]:
```
items = [0, None, 0.0, True, 0, 7]  # True and 7 evaluate to True
found = False                       # this is called "flag"

for item in items:
    print('scanning item', item)
    if item:
```

```
        found = True                # we update the flag
        break

if found:                           # we inspect the flag
    print('At least one item evaluates to True')
else:
    print('All items evaluate to False')
```

```
scanning item 0
scanning item None
scanning item 0.0
scanning item True
At least one item evaluates to True
```

1.5 else

If the loop ends normally, because of exhaustion of the iterator (**for** loop) or because the condition is finally not met (**while** loop), then the **else** suite (if present) is executed.

In case execution is interrupted by a **break** statement, the **else** clause is not executed.

[16]:
```
people = [('James', 17), ('Kirk', 9), ('Lars', 13), ('Robert', 8)]
driver = None

# old way!
for person, age in people:
    if age >= 18:
        driver = (person, age)
        break

if driver is None:
    print('Driver not found.')
```

```
Driver not found.
```

[17]:
```
# the same loop in a pythonic way
for person, age in people:
    if age >= 18:
        driver = (person, age)
        break
else:                           # <------------
    print('Driver not found.')
```

```
Driver not found.
```

Returning to the previous example

[18]:
```
items = [0, None, 0.0, True, 0, 7]  # True and 7 evaluate to True

for item in items:
```

```
    print('scanning item', item)
    if item:
        found = True                # we update the flag
        break
else:
    found = False                  # the "flag" will became false if the cycle␣
 ↪never runs the 'break' clause

if found:                          # we inspect the flag
    print('At least one item evaluates to True')
else:
    print('All items evaluate to False')
```

```
scanning item 0
scanning item None
scanning item 0.0
scanning item True
At least one item evaluates to True
```

1.6 Iterables (optional)

- An iterable is an object capable of returning its members one at a time.
- Examples of iterables include all sequence types (such as `list`, `str`, and `tuple`) and some non-sequence types like `dict`, or `file` objects.
- Objects of any classes you define with an `__iter__()` or `__getitem__()` method.

[19]:
```
for key in {"title": "And Now for Something Completely Different", "year":␣
 ↪1971}:
    print(key)
```

```
title
year
```

[20]:
```
for key, value in {"title": "And Now for Something Completely Different",␣
 ↪"year": 1971}.items():
    print(key, value)
```

```
title And Now for Something Completely Different
year 1971
```

2 Exercises

Go here...