Theory: Any and all

23 minutes 4 / 5 problems solved

By now, you certainly know that Python has a lot of different **built-in functions** that help developers work more efficiently. Built-in functions are always available, so you don't need to declare or import them. Just call such a function whenever you need it. You have already seen one of those functions, it is `print()`. Today we will learn two more built-in functions `any()` and `all()` and find out how and when to use them.

Be careful though, these functions work only with **iterable** objects, e.g. strings and lists. A list is iterable, so we will use it to illustrate the theoretical part and to show how `any()` and `all()` work.

§1. Function any()

The result of the `any()` function call is the boolean value: it returns `True` if an element or a group of elements in an iterable object are evaluated `True`. Otherwise, it returns `False`. Let's take a look at the example. Imagine that you and your friends, Jam and Andy, wrote a test and got your results in the form of a list with `True` and `False` values. The test is passed if at least one answer is correct. Now you need to check if you and your friends passed that test.

```
1 your_results = [True, False, False]
2 print(any(your_results))  # True
```

As you know, the value `True` corresponds to 1, while `False` can be represented by 0, therefore, you can replace the boolean values with the numerical ones in the list above and get the same result.

```
1 your_results = [1, 0, 0]
2 print(any(your_results))  # True
```

In fact, all numbers other than 0 will be regarded as `True`, even the negative ones. That part stems from how the bool() function works.

Now back to our test example, you passed with one correct answer. What about your friends, Jam and Andy? Andy has a different list of results to check.

```
1 andy_results = [False, False, False]
2 print(any(andy_results))  # False
```

Unfortunately, your friend Andy failed. What about Jam? Well, this friend of yours didn't write the test at all, so he got an empty list of results.

```
1 jam_results = []
2 print(any(jam_results))  # False
```

The list doesn't contain any elements, and since no `True` value is to be found the `any()` function returns `False`.

So what does the `any()` function do? First, it takes a list as an argument, then evaluates all the elements of this list to find at least one `True`, if so, it returns `True`, otherwise, the result is `False`.

§2. Function all()

The `all()` function works pretty much like `any()`. The difference is that `all()` function checks if all the elements of an iterable object are `True` and returns `True` if they are. Otherwise, you get `False`. Do you remember the story from the previous section where we checked the results of the test?

Current topic:

Any and all ...

Topic depends on:

✓ List comprehension 5★ ...

Table of contents:

Let's proceed. Imagine yet another test, this time the final one. To succeed, you should answer all the questions correctly. How did it go this time for you and the two friends of yours?

```
your_results = [True, False, False]
print(all(your_results))  # False
```

As you can see, not all the answers in your case are correct, so you didn't pass the test. What about Andy's results?

```
andy_results = [True, True, True]
print(all(andy_results))  # True
```

Luckily, Andy passed. Jam seems to have a vacation. His list of results is empty again.

```
jam_results = []
print(all(jam_results))  # True
```

The list doesn't contain any elements, but the `all()` function will return `True` because it searches for any `False` values. No `False` values result in `True`. Be careful with this scenario.

§3. Non-boolean values

Pay attention to the fact that `any()` and `all()` can take a list containing non-boolean values. Let's recall how they are evaluated.

Empty sequences, e.g. strings and lists, as well as zero, are equivalent to `False`, the same applies to the constant `None`. Non-empty sequences are equivalent to `True`.

> Be cautious, the result of calling the `all()` function on an empty sequence differs from converting an empty sequence to a boolean value. The result of `all(list())` is `True`, the result of `bool(list())` is `False`.

Here is a list with false values. `any()` and `all()` will have the same behavior in this example:

```
rocket_science_scores = [0, -0, 0.0, +0]
any(rocket_science_scores)  # False
all(rocket_science_scores)  # False
```

Now, let's look at the scores of some simpler subject:

```
math_scores = [0, 1, 2, 3]
any(math_scores)  # True
all(math_scores)  # False
```

As shown, `all()` doesn't return `True` for a list where false values are present. Consider the last case:

```
biology_scores = [1, 2, 3, 4]
any(biology_scores)  # True
all(biology_scores)  # True
```

The list `biology_scores` has no false values, that's why both functions result in `True`.

Also, you can turn the elements of your list into the boolean values via comparison. Suppose, we have a list of scores and want to check whether some are equal to 3 or greater. It can be done like this:

```
scores = [1, 2, 3, 4]

boolean_scores = [score >= 3 for score in scores]  # [False, False, True, True]
print(any(boolean_scores))  # True
print(all(boolean_scores))  # False
```

However, lists may contain different elements, e.g. strings or nested lists, in such cases, the behavior of `any()` and `all()` would depend largely on the contents. Keep that in mind.

§4. Conditions

Coders often use `any()` and `all()` functions in conditions. It helps to check the elements of iterable objects quickly and to avoid complex constructions.

Let's choose a candy box for Valentine's Day. Each box contains several types of sweets. But you are interested in the even amount of candies of each type because, obviously, you will share them with your valentine.

```
box = [10, 20, 33]

if any([candy % 2 for candy in box]):
    print("It is not a proper gift.")
else:
    print("Perfect!")
```

Short and sweet, isn't it? Life is like a box of chocolates! As long as the values you deal with can be converted to `True` and `False`, it's safe to use both functions in conditions.

§5. To sum up

We learned what `any()` and `all()` functions can do and how they work. As you can see, these functions are an efficient tool that helps check conditions and may improve the readability of your code.

Report a typo

344 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

Comments (8) Hints (1) Useful links (0) Show discussion