4.Cycles

November 29, 2018

1 Cycles

Typically there are steps in a program which you want to conduct several times
For example, here is a variant of protocol to preprocess sample: 1. add ethanol 1. evaporate 1. add internal standard 1. add derivatization agent 1. vortex

Imagine that you have 10 samples and want to process all of them. What should you do?

Here comes the cycle! This concept allows you to make something as many times as you want
In common language:

For sample in samples: 1. add ethanol 1. evaporate 1. add internal standard 1. add derivatization agent 1. vortex

In python:

```
samples = `some collection`

for sample in samples:
    cycle_body

for iteration_variable_name in iterable:
    instructions
```

```
In [2]: numbers = [1, 2, 3, 4, 5]

        for number in numbers:
            print(number)
```

```
1
2
3
4
5
```

```
In [3]: # More sophisticated variant
        for number in numbers:
            print(number * 3)
```

```
3
6
```

```
9
12
15
```

```
In [5]: # Branching can take place inside a cycle
        numbers = [1, -2, 3, 0, 5]

        for number in numbers:
            if number > 0:
                print(number, 'is positive')
            elif number < 0:
                print(number, 'is negative')
            else:
                print(number, 'is zero!')
```

```
1 is positive
-2 is negative
3 is positive
0 is zero!
5 is positive
```

1.1 continue

Now imagine that you need to skip cycle body for some values. To do it you can use `continue` key word

```
In [7]: for number in numbers:
            if number >= 0:
                continue
            elif number < 0:
                print(number, 'is negative')
```

```
-2 is negative
```

```
In [6]: # Better way to do it
        for number in numbers:
            if number >= 0:
                continue
            print(number, 'is negative')
```

```
-2 is negative
```

1.2 break

Another situation - you wanna cycle to run until it reach some value

```
In [8]: for number in numbers:
            if number != 0:
                print('Working with', number, '...')
            else:
                print('Encountering 0, exiting cycle...')
                break
```

```
Working with 1 ...
Working with -2 ...
Working with 3 ...
Encountering 0, exiting cycle...
```

Obviously you can combine them

```
In [20]: for num in numbers:
             if num % 2 == 1:
                 continue
             elif num > 2:
                 print(num)
                 break
             print(num, 'is even')
```

```
-2 is even
0 is even
```

1.3 range

There is a number of useful functions which we can use to create iterables or collection via these iterables

`range(start, stop, step)` - function-generator (later about it) which is nice to creating different ranges * `start` - start of numeric range, 0 by default * `stop` - stop of numeric range * `step` - step of range, 1 by default

```
In [21]: for i in range(0, 5, 1):
             print(i)
```

```
0
1
2
3
4
```

```
In [22]: # Equivalent to
         for i in range(5):
             print(i)
```

```
0
1
2
3
4
```

```
In [23]: # Changing the range
         for num in range(3, 10):
             print(num)
```

```
3
4
5
6
7
8
9
```

```
In [25]: # With other step
         for num in range(3, 10, 3):
             print(num)
```

```
3
6
9
```

```
In [27]: # Backwards
         for num in range(10, 3, -1):
             print(num)
```

```
10
9
8
7
6
5
4
```

1.4 while

Also there is another type of cycle - `while`

*Shake falcon **while** there is no phase separation!*

```
In [28]: i = 5
```

```
        while i > 0:
            print(i)
            i = i - 1
```

```
5
4
3
2
1
```

```
In [ ]: while predicate:
            cycle_body
```

```
In [30]: a = '!'

         while a != 'X':
             print('a is not X!')
             a = 'X'
```

```
a is not X!
```

```
In [31]: p = 35 > 17

         while p:
             print('Ye, 35 is greater than 17')
             print('And it will not change in next iteration')
             print('What are you expecting from this code cell?')
             print('If there will be no break')
             break
```

```
Ye, 35 is greater than 17
And it will not change in next iteration
What are you expecting from this code cell?
If there will be no break
```

“‘python p = True
while True: ... ad infinitum ...“‘

You should make change to your variables which will make your predicate `False` or have a reachable `break` keyword somewhere in your cycle

1.5 2 common ways to iterate over list elements

Preferred and should be used whenever possible, pythonic way `python for element in my_list:  do something with element`

Bad style `python for i in range(len(my_list)):  do something with my_list[i]`

It is possible that you will need index of element in addition to element per se
Use awesome `enumerate()` in this case

```
   python for i, element in enumerate(my_list):     do something with element
do something with i
```

```
In [33]: for i, element in enumerate(numbers):
             print(element, 'is on index', i, 'in list')
```

```
1 is on index 0 in list
-2 is on index 1 in list
3 is on index 2 in list
0 is on index 3 in list
5 is on index 4 in list
```

You can iterate over all built-in data types which are collections. Everything which can be used in iteration is iterable.

Almost everything except numbers and booleans is iterable (strings too)

1.6 Awesome shortcuts for arithmetic operations

It is quite common to update variables like `i = i - 1` in example above. It is too long to write so, thus we have better way

Shortcuts: * `i = i + x` `i += x`

- `i = i - x` `i -= x`
- `i = i * x` `i *= x`
- `i = i / x` `i /= x`
- `i = i // x` `i //= x`
- `i = i % x` `i %= x`
- `i = i ** x` `i **= x`

```
In [ ]:
```