**Peephole Optimization in Python**

In Python, **Peephole optimization** is a code optimization technique that occurs at compile time for better performance & speed of your code.

In Peephole optimization, Python optimizes code either by pre-calculating constant expressions or by membership tests.

These optimizations occur behind the scenes. Normally you don't even notice that all these stuffs are happening.

To understand peephole optimizations, let's start from constant expressions first and then we move to membership test.

**Constant Expressions**

Suppose in your code, you are using number of seconds in a week like seconds\_in\_week = 7 \* 24 \* 60 \* 60. If this statement is repeated many times during the lifetime of your program then this expressions needs to be calculated again and again. Which certainly decreases performance.

As a developer, you may think that if that decreases performance, I will write seconds\_in\_week = 604800. However, using value 604800 instead of 7 \* 24 \* 60 \* 60 is not much readable for this purpose. So, from a readability view point, it is messed up right?

**But don't worry! Python pre-calculates constant expressions 7 \* 24 \* 60 \* 60 and replace it for 604800. So, you can write 7 \* 24 \* 60 \* 60 as many time as required without degrading performance.**

**So, what gets pre-calculated?**

Python pre-calculates:

1. All constants like 7 \* 24 \* 60 \* 60
2. All constant lists, tuples, sets & strings whose length is less than 20 **(This number can be different for different Python implementation)**.  
     
   For example (1, 2) \* 6 i.e. (1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2) gets precalculated because its total length is 12 but (1, 2) \* 12 because its total length is 24.

To verify pre-calculation of constants, consider following code and output

**def** constant\_expression\_test():

seconds\_in\_week = 7 \* 24 \* 60 \* 60

short\_strings = "abc" \* 6

tuples = (1,2) \* 4

long\_strings = "very very long sentence" \* 80

**print**(constant\_expression\_test.\_\_code\_\_.co\_consts)

Output

(None,

604800,

'abcabcabcabcabcabc',

(1, 2, 1, 2, 1, 2, 1, 2),

'very very long sentence',

80)

**Note:**constant\_expression\_test.\_\_code\_\_.co\_consts gives all constants used inside the function constant\_expression\_test

In the above output, we see that 7 \* 24 \* 60 \* 60, "abc" \* 6 and (1,2) \* 4 are pre-calculated but not "very very long sentence" \* 80.

**Membership Tests**

Another optimzation that Python does out of the box is **membership test** by converting mutable data structures to immutable data structures.

Python does this mutable to immutable conversion because immutable data structures are relatively faster than mutable.

So in membership test, constant lists are converted to tuples, constant sets are converted to frozensets and so on.

Consider following example to illustrate the concept of membership test:

**def** membership\_test():

**for** name **in** ["Jack", "John", "Dev", "Saki"]:

**pass**

**for** a **in** {1,2,3,4}:

**pass**

**print**(membership\_test.\_\_code\_\_.co\_consts)

Output

(None, ('Jack', 'John', 'Dev', 'Saki'), frozenset({1, 2, 3, 4}))

Here list ["Jack", "John", "Dev", "Saki"] is converted to tuple ('Jack', 'John', 'Dev', 'Saki') and similarly set {1,2,3,4} is converetd to frozenset({1, 2, 3, 4}).

Reference [Peephole Optimization in Python (codesansar.com)](https://www.codesansar.com/python-programming/peephole-optimization.htm)