Homework03 for Architecture.

Files:

- **main.py(2KB)**: main function which reads, writes, etc...
- **container.py(2KB)**: container with all the functions.
- **__init__.py(0B)**: makes matrices directory be seen as a module.
- **baseMatrix.py(96B)**: basic matrix structure with all the functions.
- **matrix.py(1KB)**: usual matrix structure with all the functions.
- **diagonalMatrix.py(1KB)**: diagonal matrix structure with all the functions.
- **lowerTriangularMatrix.py(1KB)**: lower-triangular matrix structure with all the functions.
- **FillingScript.py(1KB)**: script which fills tests directory with random tests.

Used hardware and software:

- OS: MacOS Big Sur 11.6
- RAM: 16gb
- Architecture: x86-x64
- IDE: PyCharm

Command line input guide:

1) Write ***./ArchitectureHW3 -f [inputFileName].txt [outputFileName].txt*** for the file input. 2) Write ***./ArchitectureHW3 -n [number] [outputFileName].txt*** for the random input.

File input guide:

You need to input a couple of matrices according to this template:

1) Input type: ***0*** for usual matrix, ***1*** for diagonal matrix, ***2*** for lower-triangular matrix. 2) Input size: ***N*** for ***NxN*** matrix. 3) Input ***N^2*** real numbers for usual matrix, ***N*** real numbers for diagonal matrix and ***N * (N + 1) / 2*** real numbers for lower-triangular matrix.

Tests:

For your conviniece I created a set of 11 tests via python *FillingScript.py* with input00.txt as an empty file.

Average time:

- for 1 matrix: 3ms
- for 10 matrices: 42ms
- for 100 matrices: 1746ms
- for 1000 matrices: 131180ms

Comparing to procedure and object-oriented programming:

Dynamic type programming gives more freedom, however you will pay quite a high cost for it: firstly, the program becomes rather slow, secondly, it is much more harder to catch bugs.

Number of matrices	No OOP C++	OOP C++	Python
1	0ms	2ms	3ms, 7.32MB
10	2ms	17ms	42ms, 8.84MB
100	62ms	246ms	1746ms, 26.82MB
1000	4056ms	3338ms	131180ms, 165.44MB

As we can see from the table, it is clear that interpreted dynamically typed languages are ***much slower*** than the statically typed ones. Also without Object Oriented programming code works quite faster, however on 1000 matrices C-style C++ is slower because in first homework I used fstream.h instead of stdio.h. From my own experience, I know that python uses ***more memory*** because, firstly, it is interpreter, secondly, it has a garbage collector while in C/C++ we delete unused memory with our own hands and, thirdly, collections like lists and dictionaries use more memory and work slower than usual arrays.

How variables are stored in python:

In python there are two different types of simple data: mutable (list, dict, set...) and immutable (int, string, tuple...). Basically, all variables in python are actually pointers for some place in pc memory. However, there are some differences between these two types, let me demonstrate it with some simple code:

Mutable:

```
 a = 1
print('a id:', id(a))
```

> a id: 4427782848

```
 b = 1
print('b id:', id(b))
```

> b id: 4427782848

```
 a += 1
print('a id:', id(a))
```

> a id: 4427782880

Immutable:

```
 a = [0, 1, 2]
print('a id:', id(a))
```

> a id: 140546841487424

```
 b = [0, 1, 2]
print('b id:', id(b))
```

> b id: 140546841493952

```
 a.append(3)
print('a id:', id(a))
```

> a id: 140546841487424

For the memory description look at MemoryDescriptiom.pdf:

https://github.com/OFFLUCK/Architecture/blob/master/ArchitectureHW3/Description/MemoryDescription.pdf

Conclusion:

The time now doesn't feel like linear, it is more like O(L * M * N), L - number of matrices, M and N - their sizes. Interpreted dynamic programming languages are much slower and harder to debug than static ones, yet faster and easier to write.