Coding Test – David Viktora

1.

Result

```
1
3
```

Explanation

- in JavaScript, variables defined inside of the function exist only inside the context of that function.
- Control sequences do not have their own context.

2.

Result

```
7
8
```

Explanation

- Since we assign instance of `foo` to `moo`, calling that instance multiple times will use the same instance of variable `baz`. Moving declaration of `baz` to the main function will have the same effect, but make the code more readable.

```
(function() {
  var baz = 3;
  function foo(x) {
    return function (y) {
      console.log(x + y + (++baz));
    }
  }
  var moo = foo(2);
  moo(1); // What is output here?
  moo(1); // What is output here?
})();
```

In case we wanted to keep baz inside foo, but avoid multiple calls affecting each other, we would need multiple instances of foo:

```
  var moo = foo;
  moo(2)(1);
  moo(2)(1);
```

3.

```
3.py
```

4.

```
4.c
```

5.

Result

```
list1 = [10, 'a']
list2 = [123]
list3 = [10, 'a']
```

Explanation

In Python, mutable default arguments are created only once, after the function is defined. In this example, empty list is created and values are appended to it in the first and third executions.

6.

Result

```
[6, 6, 6, 6]
```

Explanation

In Python closures, non-local variables are accessed in the time of execution, not while the closure is defined. In this case, in the time of execution, `i` in the function's outer scope is always equal to `3` (last value in `range(4)`) in this case.

7.

Result

```
DFS - ABEIJFCDGKLH
BFS - ABCDEFGHIJKL
```

Explanation

Depth First Search – starting at the root node, the (for example) leftmost node is always visited, going as deep as possible. Whenever the node has no children, we go back up, searching for the first node with unvisited child nodes.

Breath First Search – all nodes with the same distance from the root node are reached before proceeding to the next level.

8.

Result

```
1 cycle in graph - ABC
```

Explanation

We can use DFS graph traversal to find cycles in a graph. Whenever we reach node that was visited before, a new cycle has been found.

9.

SMILES specification/notation. DFS is used to generate the string, any cycles are broken to create a spanning tree. The problem with this notation is the fact, that one molecule can be represented with multiple strings, based on:

- selection of starting node
- order in which branches are listed
- way in which cycles are broken

Molecule in this task could be written for example as:

- CC(N1CCCC1)(C)C2=CC=CC=C2
- CC(C)(N1CCCC1)C2=CC=CC=C2
- C1=CC=C(C=C1)(C(C)(C)(N1CCCCC1))
- ….

10.

- naive solution

Test floors divisible by three – 3, 6, 9… . If the egg does not break, go 3 floors up. If it breaks, drop the second egg from i-1th floor. If the second egg breaks too, the result is i-2, otherwise the result is i-1.

Worst case scenario: 34 drops, 2 broken eggs

- improved solution

Divide floors to x sections. Always test the first floor of the section, skipping the first section. If the egg breaks, test the previous section, floor by floor. The best possible way to divide floors to sections is described by this formula:

```
n + (n-1) + (n-2) + … = 100
```

This way, the higher section we test, the number of subsequent floor by floor tests will be lower. Calculating the size of first section:

```
n * (n+1) / 2 = 100
n = 13.651 ≅ 14
```

That way, we split floors to sections starting at floors 14, 27, 39, 50, 60, 69, 77, 84, 90, 95, 99, 100. We test each of these floors. If the egg breaks, we test all floors in the previous section, one by one.

Worst case scenario: 14 drops, 2 broken eggs. This can happen for multiple results, e.g. floors 13, 26, 38, .. = last floor in every section.

- infinite eggs – binary search

Worst case scenario: 7 drops, 6 broken eggs

11.

“Knapsack problem”

- in this case, the solution is simple and can be found manually.

We calculate value per megabyte for every file:

File	1	2	3	4	5	6
Size	1	3	4	5	6	10
Value	15	9	5	10	26	40
Value per MB	15	3	5/4	2	13/3	4

Now we add files with highest “performance” (value per MB) until we reach maximum size:

Added file	1	5	2	
Current size	1	7	10	
Current value	15	41	50	

This way, we see we can save files 1, 5 and 2 with 50$ total value.

- in other cases, the solution may not be that clear, mostly because of the possibility of underfilling the “bag” - sometimes using files with lower value per MB will lead to higher total value, since we will use bigger part of the bag’s capacity.

Following algorithms can be used:

- brute force – all combinations are measured

- branch and bounds – improved brute force approach, solutions that cannot be better then current best are not calculated

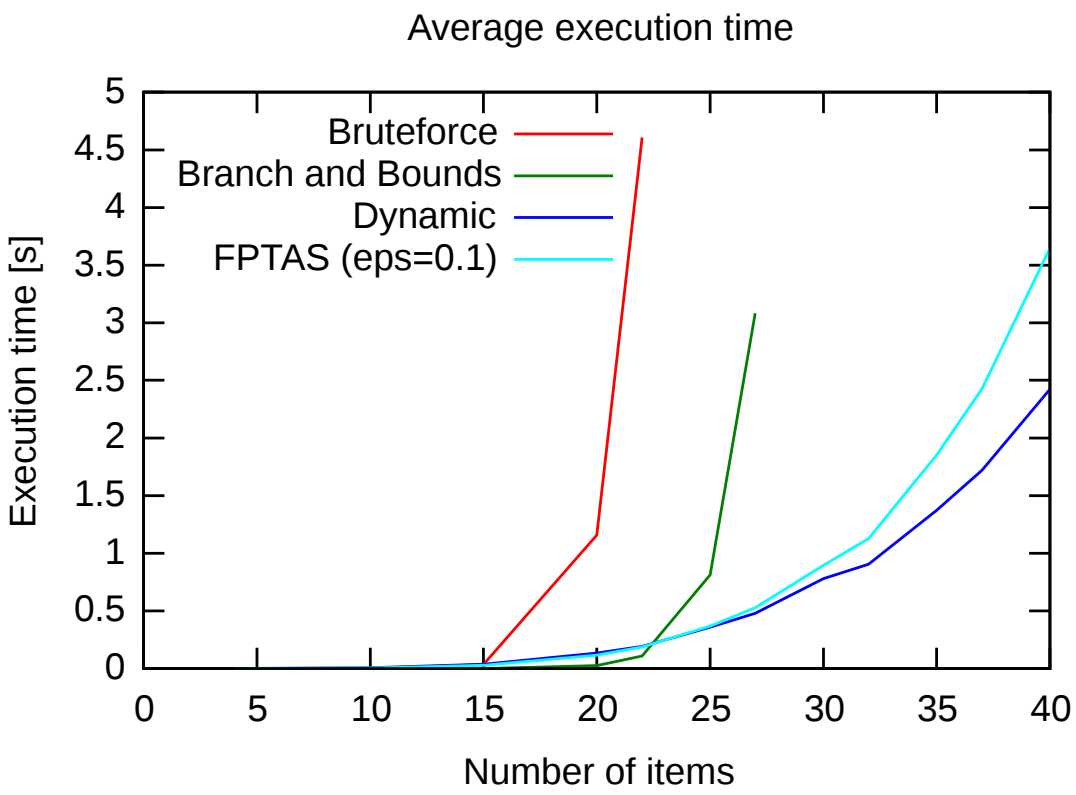
- Dynamic programming

- FPTAS (result is not always optimal)

I implemented algorithms above in Python as part of “Problems & Algorithms” course at FIT CTU.

Source codes and results: https://github.com/Fanarim/MI-PAA

Performance chart of my implementation:



12.

Usage:

```
# go to the program's directory
 cd 12/number_echoer/

 # Generate message catalogs:
 for LANGUAGE in `ls locale/ | grep -v .pot`; do msgfmt \
 locale/$LANGUAGE/LC_MESSAGES/number_echoer.po -o \
 locale/$LANGUAGE/LC_MESSAGES/number_echoer.mo; done;

 # Run the program in selected language:
 LANG=cs_CZ.UTF-8 ./number_echoer.py 12345
 LANG=de_DE.UTF-8 ./number_echoer.py 12345
 LANG=en_US.UTF-8 ./number_echoer.py 12345
```

See `12/TODO` for list of issues and possible fixes and improvements.