Задание 1.
Исходные данные:
Подсчитать, сколько было выделено памяти под переменные в ранее разработанных программах в рамках первых трех уроков. Проанализировать результат и определить программы с наиболее эффективным использованием памяти.

```
a = 50
b = '+', '-', '*', '/', '0'
c = 75
def func(a, b, c):
    try:
        a = int(input("Введите число: "))
        b = input("Введите математический знак действия: ")
        c = int(input("Введите число: "))
    except zerodivisionerror:
        return
    if b==0:
        print("Программа завершила свою работу")
    elif c==0:
        print("Делить на ноль нельзя")
    else:
        a = int(input("Введите число: "))
        b = input("Введите математический знак действия: ")
        c = int(input("Введите число: "))
    d = a + c
    e = a - c
    f = a * c
    g = a / c
    s = d, e, f, g
    return s

print(func(a, b, c))
```

Решение:

```
Python 3.8.10 (default, Jun  2 2021, 10:49:15)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import memory_profiler
>>> from memory_profiler import profile
>>> def func(a, b, c):
...     try:
...         a = int(input("Введите число: "))
...         b = input("Введите математический знак действия: ")
...         c = int(input("Введите число: "))
...     except zerodivisionerror:
...         return
...     if b==0:
...         print("Программа завершила свою работу")
...     elif c==0:
...         print("Делить на ноль нельзя")
...     else:
...         a = int(input("Введите число: "))
...         b = input("Введите математический знак действия: ")
...         c = int(input("Введите число: "))
...     d = a + c
...     e = a - c
...     f = a * c
...     g = a / c
...     s = d, e, f, g
...     return s
...     print(func(a, b, c))
...     if __name__ == '__main__':
...         func(a, b, c)
...
>>> import sys
>>> sys._debugmallocstats()
Small block threshold = 512, in 32 size classes.

class   size   num pools   blocks in use  avail blocks
-----   ----   ---------   -------------  ------------
    0     16           5          1019           246
    1     32          51          6381            45
    2     48         355         29712           108
    3     64        1217         76609            62
    4     80         735         36718            32
    5     96         258         10830             6
    6    112         152          5451            21
    7    128         105          3243            12
    8    144         491         13742             6
    9    160          55          1361            14
   10    176         800         18373            27
   11    192          38           785            13
   12    208          35           649            16
   13    224          63          1127             7
   14    240          28           441             7
   15    256          25           361            14
   16    272          25           338            12
   17    288          18           251             1
   18    304         119          1533            14
   19    320          18           205            11
   20    336          16           182            10
   21    352          16           167             9
   22    368          14           145             9
   23    384          15           140            10
   24    400          17           164             6
   25    416          23           203             4
   26    432          33           280            17
   27    448          25           210            15
   28    464          22           174             2
   29    480          20           154             6
   30    496          24           168            24
   31    512          32           213            11

# arenas allocated total           =                  136
# arenas reclaimed                 =                   60
# arenas highwater mark            =                   76
# arenas allocated current          =                   76
76 arenas * 262144 bytes/arena      =           19,922,944

# bytes in allocated blocks        =           19,354,096
# bytes in available blocks        =              106,544
14 unused pools * 4096 bytes       =               57,344
# bytes lost to pool headers       =              232,800
# bytes lost to quantization       =              172,160
# bytes lost to arena alignment    =                    0
Total                              =           19,922,944

       2 free PyCFunctionObjects * 56 bytes each =                  112
           16 free PyDictObjects * 48 bytes each =                  768
           4 free PyFloatObjects * 24 bytes each =                   96
          1 free PyFrameObjects * 368 bytes each =                  368
           65 free PyListObjects * 40 bytes each =                2,600
         30 free PyMethodObjects * 48 bytes each =                1,440
   7 free 1-sized PyTupleObjects * 32 bytes each =                  224
  17 free 2-sized PyTupleObjects * 40 bytes each =                  680
   5 free 3-sized PyTupleObjects * 48 bytes each =                  240
   3 free 4-sized PyTupleObjects * 56 bytes each =                  168
   3 free 5-sized PyTupleObjects * 64 bytes each =                  192
   1 free 6-sized PyTupleObjects * 72 bytes each =                   72
   5 free 7-sized PyTupleObjects * 80 bytes each =                  400
   2 free 8-sized PyTupleObjects * 88 bytes each =                  176
   3 free 9-sized PyTupleObjects * 96 bytes each =                  288
 0 free 10-sized PyTupleObjects * 104 bytes each =                    0
 6 free 11-sized PyTupleObjects * 112 bytes each =                  672
 1 free 12-sized PyTupleObjects * 120 bytes each =                  120
 1 free 13-sized PyTupleObjects * 128 bytes each =                  128
 3 free 14-sized PyTupleObjects * 136 bytes each =                  408
 1 free 15-sized PyTupleObjects * 144 bytes each =                  144
 3 free 16-sized PyTupleObjects * 152 bytes each =                  456
 0 free 17-sized PyTupleObjects * 160 bytes each =                    0
 0 free 18-sized PyTupleObjects * 168 bytes each =                    0
 4 free 19-sized PyTupleObjects * 176 bytes each =                  704
>>>
```

# python x64 bit, os ubuntu20.04ltse x64 bit.

Исходные данные:

```
import random
from random import randrange
randrange(1, 101)
n = round(randrange(1, 101))
i = 1
print("Я загадал число. Для его отгадывания есть 10 попыток.")
while i <= 10:
    a = int(input("Введите число: "))
    if a>n:
        print("Число больше загаданного")
    elif a <n:
        print("Число меньше загаданного")
    else:
        print("Вы угадали число")
        break
    i += 1
else:
    print("Вы проиграли. Я загадал вот такое число:", n)
```
Решение:
```
Python 3.8.10 (default, Jun  2 2021, 10:49:15)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import memory_profiler
>>> from memory_profiler import profile
>>> import random
>>> from random import randrange
>>> randrange(1, 101)
>>> n = round(randrange(1, 101))
>>> i = 1
>>> print("Я загадал число. Для его отгадывания есть 10 попыток.")
>>> while i <= 10:
...     a = int(input("Введите число: "))
...     if a>n:
...         print("Число больше загаданного")
...     elif a <n:
...         print("Число меньше загаданного")
...     else:
...         print("Вы угадали число")
...         break
...     i += 1
... else:
...     print("Вы проиграли. Я загадал вот такое число:" n)
...     if __name__ == '__main__':
...         func(n)
...
Введите число: 75
Число меньше загаданного
Введите число: 88
Число меньше загаданного
Введите число: 99
Число больше загаданного
Введите число: 95
Число меньше загаданного
Введите число: 96
Число меньше загаданного
Введите число: 97
Число меньше загаданного
Введите число: 98
Вы угадали число
>>> import sys
>>> sys._debugmallocstats()
Small block threshold = 512, in 32 size classes.

class   size   num pools   blocks in use  avail blocks
-----   ----   ---------   -------------  ------------
```

```
    0    16       5      1019     246
    1    32      51      6381      45
    2    48     355     29712     108
    3    64    1217     76609      62
    4    80     735     36719      31
    5    96     259     10830      48
    6   112     152      5451      21
    7   128     105      3241      14
    8   144     491     13740       8
    9   160      55      1360      15
   10   176     800     18372      28
   11   192      38       786      12
   12   208      35       649      16
   13   224      63      1126       8
   14   240      28       441       7
   15   256      25       361      14
   16   272      25       338      12
   17   288      18       251       1
   18   304     119      1533      14
   19   320      18       205      11
   20   336      16       182      10
   21   352      16       167       9
   22   368      14       145       9
   23   384      15       140      10
   24   400      17       165       5
   25   416      23       203       4
   26   432      33       279      18
   27   448      25       210      15
   28   464      22       174       2
   29   480      20       154       6
   30   496      24       168      24
   31   512      32       214      10

# arenas allocated total           =              146
# arenas reclaimed                 =               70
# arenas highwater mark            =               76
# arenas allocated current         =               76
76 arenas * 262144 bytes/arena     =       19,922,944

# bytes in allocated blocks        =       19,353,744
# bytes in available blocks        =          110,928
13 unused pools * 4096 bytes       =           53,248
# bytes lost to pool headers       =          232,848
# bytes lost to quantization       =          172,176
# bytes lost to arena alignment    =                0
Total                              =       19,922,944

       2 free PyCFunctionObjects * 56 bytes each =              112
          16 free PyDictObjects * 48 bytes each =               768
           4 free PyFloatObjects * 24 bytes each =               96
         0 free PyFrameObjects * 368 bytes each =                 0
          65 free PyListObjects * 40 bytes each =             2,600
       30 free PyMethodObjects * 48 bytes each =               1,440
   7 free 1-sized PyTupleObjects * 32 bytes each =                224
  17 free 2-sized PyTupleObjects * 40 bytes each =                680
   5 free 3-sized PyTupleObjects * 48 bytes each =                240
   3 free 4-sized PyTupleObjects * 56 bytes each =                168
   3 free 5-sized PyTupleObjects * 64 bytes each =                192
   1 free 6-sized PyTupleObjects * 72 bytes each =                 72
   6 free 7-sized PyTupleObjects * 80 bytes each =                480
   3 free 8-sized PyTupleObjects * 88 bytes each =                264
   3 free 9-sized PyTupleObjects * 96 bytes each =                288
 0 free 10-sized PyTupleObjects * 104 bytes each =                  0
 6 free 11-sized PyTupleObjects * 112 bytes each =                672
 1 free 12-sized PyTupleObjects * 120 bytes each =                120
 1 free 13-sized PyTupleObjects * 128 bytes each =                128
 3 free 14-sized PyTupleObjects * 136 bytes each =                408
 1 free 15-sized PyTupleObjects * 144 bytes each =                144
 3 free 16-sized PyTupleObjects * 152 bytes each =                456
 0 free 17-sized PyTupleObjects * 160 bytes each =                  0
 0 free 18-sized PyTupleObjects * 168 bytes each =                  0
 4 free 19-sized PyTupleObjects * 176 bytes each =                704
>>>
```

```
# python x64 bit, os ubuntu20.04ltse x64 bit
```

Исходные данные:

```
a = 6
b = 6
c = 6
def treug(a, b, c):
    try:
        a = int(input("Введите число: "))
        b = int(input("Введите число: "))
        c = int(input("Введите число: "))
    except zerodivisionerror:
        return
    if a+b>c:
        print('Равнобедренный')
    elif a+b==c:
        print('Равносторонний')
    else:
        a + b < c
        print('Разносторонний')

print(treug(a, b, c))
```

Решение:

```
Python 3.8.10 (default, Jun  2 2021, 10:49:15)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import memory_profiler
>>> from memory_profiler import profile
>>> def treug(a, b, c):
...     try:
...         a = int(input("Введите число: "))
...         b = int(input("Введите число: "))
...         c = int(input("Введите число: "))
...     except zerodivisionerror:
...         return
...     if a+b>c:
...         print('Равнобедренный')
...     elif a+b==c:
...         print('Равносторонний')
...     else:
...         a + b < c
...         print('Разносторонний')
...     print(treug(a, b, c))
...     if __name__ == '__main__':
...         treug(a, b, c)
...
>>> import sys
>>> sys._debugmallocstats()
Small block threshold = 512, in 32 size classes.

class   size   num pools   blocks in use  avail blocks
-----   ----   ---------   -------------  ------------
    0     16        5          1019          246
    1     32       51          6381           45
    2     48      355         29712          108
    3     64     1217         76604           67
    4     80      735         36721           29
    5     96      258         10832            4
    6    112      152          5454           18
    7    128      105          3242           13
    8    144      491         13741            7
    9    160       55          1360           15
   10    176      800         18373           27
   11    192       38           786           12
   12    208       35           651           14
   13    224       63          1126            8
   14    240       28           441            7
   15    256       25           361           14
   16    272       25           338           12
   17    288       18           251            1
   18    304      119          1533           14
   19    320       18           205           11
   20    336       16           182           10
   21    352       16           167            9
   22    368       14           145            9
   23    384       15           140           10
   24    400       17           164            6
   25    416       23           203            4
   26    432       33           280           17
   27    448       25           210           15
   28    464       22           174            2
   29    480       20           154            6
   30    496       24           168           24
   31    512       32           213           11
```

```
# arenas allocated total           =                 132
# arenas reclaimed                 =                  56
# arenas highwater mark            =                  76
# arenas allocated current         =                  76
76 arenas * 262144 bytes/arena     =          19,922,944

# bytes in allocated blocks        =          19,354,496
# bytes in available blocks        =             106,144
14 unused pools * 4096 bytes       =              57,344
# bytes lost to pool headers       =             232,800
# bytes lost to quantization       =             172,160
# bytes lost to arena alignment    =                   0
Total                              =          19,922,944

      2 free PyCFunctionObjects * 56 bytes each =                112
          16 free PyDictObjects * 48 bytes each =                768
          4 free PyFloatObjects * 24 bytes each =                 96
         1 free PyFrameObjects * 368 bytes each =                368
          65 free PyListObjects * 40 bytes each =              2,600
        30 free PyMethodObjects * 48 bytes each =              1,440
   7 free 1-sized PyTupleObjects * 32 bytes each =               224
  17 free 2-sized PyTupleObjects * 40 bytes each =               680
   4 free 3-sized PyTupleObjects * 48 bytes each =               192
   3 free 4-sized PyTupleObjects * 56 bytes each =               168
   3 free 5-sized PyTupleObjects * 64 bytes each =               192
   2 free 6-sized PyTupleObjects * 72 bytes each =               144
   6 free 7-sized PyTupleObjects * 80 bytes each =               480
   3 free 8-sized PyTupleObjects * 88 bytes each =               264
   3 free 9-sized PyTupleObjects * 96 bytes each =               288
 0 free 10-sized PyTupleObjects * 104 bytes each =                 0
 6 free 11-sized PyTupleObjects * 112 bytes each =               672
 1 free 12-sized PyTupleObjects * 120 bytes each =               120
 1 free 13-sized PyTupleObjects * 128 bytes each =               128
 3 free 14-sized PyTupleObjects * 136 bytes each =               408
 1 free 15-sized PyTupleObjects * 144 bytes each =               144
 3 free 16-sized PyTupleObjects * 152 bytes each =               456
 0 free 17-sized PyTupleObjects * 160 bytes each =                 0
 0 free 18-sized PyTupleObjects * 168 bytes each =                 0
 4 free 19-sized PyTupleObjects * 176 bytes each =               704
>>>
# python x64 bit, os ubuntu20.04ltse x64 bit
```

Из данных трёх программ самая лёгкая и самая экономичная по памяти оказалась вторая.