GEO1000-2020 EXAM

Delft University of Technology
GEO1000-2020 – **EXAM**

SURNAME: Veselý

NAME: Ondrej

STUDENT ID: 5162130

STUDY PROGRAMME: MSc Geomatics

FRAUD AND PLAGIARISM AWARENESS STATEMENT

I confirm herewith that:

- I made this exam alone without help from anybody else
- I have not helped any other students during the whole length of the examination time
- I have read and I am aware of the TU Delft rules (and consequences) regarding fraud and plagiarism, as written at: https://www.tudelft.nl/en/student/legal-position/fraud-plagiarism/

Question 1.

a. True
b. True
c. True
d. False
e. True

Question 2.

a.

1. Comments
2. Variable/value assignment
3. Equality checks
4. Itself
5. Termination
6. Be able to reach

b.

1. Both print and return
2. Print
3. Return
4. Print

Question 3.

First fragment will execute first branch if x == 5 is True and second branch if x == 5 is False.

Second fragment will execute both branches if x == 5 is True and second branch x == 5 is False.

Question 4.

a. [2,3]
b. True
c. 1, 2, 4, 5 (*newlines omitted)*
d. 10
e. BlueRed

Question 5.

If we leave out the default argument, the enumerate() argument *start* will default to 0, generating a list of nested tuples enumerated from 0:

[(0, (‘BSc, ‘Geography’)), (1, (‘MSc’, ‘Geomatics’,))]

Printing:

0 (‘BSc, ‘Geography’)
1 (‘MSc’, ‘Geomatics’,)

Question 6.

Default argument object are binded at function definition to the function object.

There is therefore a single list object, binded as default argument value to the *mylist_adder* function object. Since lists are mutable, we can still manipulate that single object later, which can lead to all sorts of 'unexpected' behaviour like this if we do so.

Question 7.

a. print(food[2])
b. for i in friends:
   print(i)
c. for i in L:
   print(i[1])
d. total = 0
   for p in prices:
   total += p
   print(total)

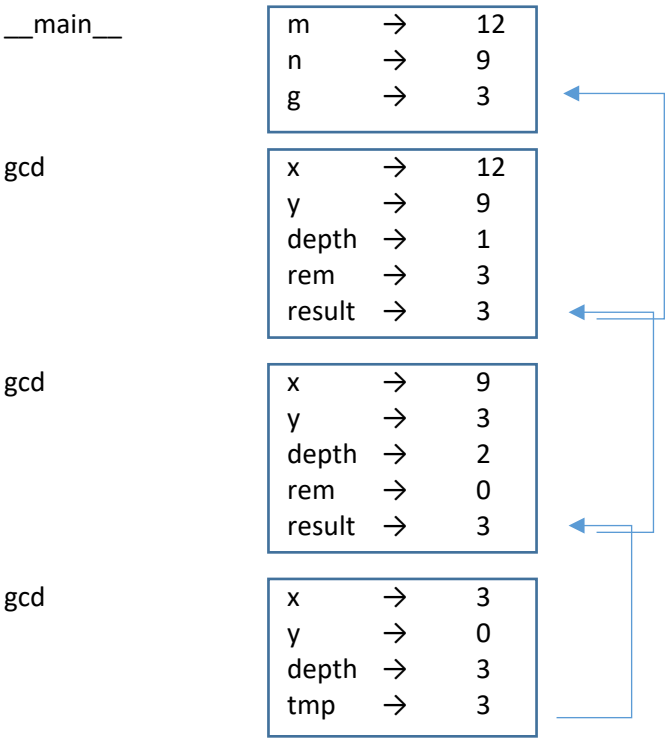
Question 8.

```
def translate(string, delimiter='-'):
  """Translate a series of digits, given as string, to their spoken English counterpart"""
  words = {0:'zero', 1:'one', 2:'two', 3:'three', 4:'four', 5:'five', 6:'six', 7:'seven', 8:'eight', 9:'nine'}
  return delimiter.join([words[int(c)] for c in string if c.isdigit()])
```

Question 9.

a. sum(plays['artist A'].values())
b. plays['artist A']['track B'] += 1
c. plays['artist C'] = {'track E': 0}
d. plays.pop('artist C')
e. plays['artist B']['track D_NEW'] = plays['artist B'].pop('track D')

Question 10.



Question 11.

```
def size_recu(t):
    left = size_recu(t.left) if t.left else 0
    right = size_recu(t.right) if t.right else 0
    return 1 + left + right
```

Question 12.

```
class Point2:

   def __init__(self, x, y):
      try:
         self.x = float(x)
         self.y = float(y)
      except:
         raise TypeError("Couldn't cast to float.")

   def cab_distance(self, other):
      if isinstance(other, Point2):
         return abs(self.x-other.x) + abs(self.y - other.y)
      else:
         return abs(self.x-other[0]) + abs(self.y - other[1])

   def __str__(self):
      return "%.4f, %.4f" % (self.x, self.y)


p, q = Point2(1,1), Point2(5,6)
print( p.cab_distance(q) )
```

Question 13.

```
class Batch:

    def __init__(self, index):
        self.index = index
        self.deliveries = {}

    def total(self):
        return sum(self.deliveries.values())

    def __str__(self):
        header = "## Batch %d - %d items" % (self.index, self.total())
        addresses = ["- %s: %d" % (k, v) for k, v in self.deliveries.items()]
        return '\n'.join([header] + addresses)

class BatchList:

    def __init__(self, capacity=7):
        self.capacity = capacity
        self.batches = [Batch(1)]

    def add(self, address, amount):
        batch = self.batches[-1]
        if self.capacity - batch.total() >= amount:
            batch.deliveries[address] = amount
        else:
            fit = self.capacity - batch.total()
            batch.deliveries[address] = fit
            self.batches.append(Batch(batch.index + 1))
            self.add(address, amount - fit)

    def __str__(self):
        return "Toiletpaper delivery" + '\n\n' + '\n\n'.join([str(b) for b in self.batches])

def main():
    orders = [
        ("Address 1", 4),
        ("Address 2", 15),
        ("Address 3", 3),
        ("Address 4", 8)
    ]

    batches = BatchList(capacity=7)
    for order in orders:
        batches.add(order[0], order[1])
    print(batches)


if __name__ == "__main__":
    main()
```