|  |  |
| --- | --- |
| **GEO1000-2020**  **EXAM** | https://upload.wikimedia.org/wikipedia/it/0/03/TUDelftLogo.png |

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.**

1. True
2. True
3. True
4. False
5. 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.**

1. [2,3]
2. True
3. 1, 2, 4, 5 (*newlines omitted)*
4. 10
5. 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.**

1. print(food[2])
2. for i in friends:

print(i)

1. for i in L:

print(i[1])

1. 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.**

1. sum(plays['artist A'].values())
2. plays[‘artist A’][‘track B’] += 1
3. plays['artist C'] = {'track E': 0}
4. plays.pop('artist C')
5. plays['artist B']['track D\_NEW'] = plays['artist B'].pop('track D')

**Question 10.**

\_\_main\_\_ m → 12

n → 9

g → 3

gcd x → 12

y → 9

depth → 1

rem → 3

result → 3

gcd x → 9

y → 3

depth → 2

rem → 0

result → 3

gcd x → 3

y → 0

depth → 3

tmp → 3

**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()