Delft University of Technology
Faculty of Architecture and the Built Environment

EXAM — GEO1000, PYTHON PROGRAMMING FOR GEOMATICS (5 EC)

Wednesday, November 4, 2020, 09h00 - 12h00
Online

Responsible teacher: dr.ir. B. M. Meijers

In case you have problems during the exam you can reach me via e-mail:
b.m.meijers@tudelft.nl
or via telephone:
015 - 27 856 42
(which will be forwarded to my mobile phone)

Introduction

This exam has 13 questions. This document has 15 pages.

The weights of the questions are stated with the questions (the total score adds up to 100 points).

Question:	1	2	3	4	5	6	7	8	9	10	11	12	13	Total
Points:	5	10	5	5	5	5	5	10	10	5	10	10	15	100

You *must* answer the questions *alone*, *without help* from anybody else. Additionally, you *must not help other students* during the whole length of the examination time.

This exam is open-book, i.e. you are allowed to use:

- The book: Think Python: How to Think Like a Computer Scientist (2nd Edition, Version 2.4.0) available at:
  `http://greenteapress.com/thinkpython2/thinkpython2.pdf`
- The lecture slides that are available on Brightspace for GEO1000, 2020-2021;
- A Python installation installed on your *own* computer and text editor / IDE. When you are asked to write a Python function / program you can thus use it, but be aware that time for the exam is limited.

Use of any other material is prohibited and be aware of the TU Delft rules (and consequences) regarding fraud and plagiarism. They can be found here:
`https://www.tudelft.nl/en/student/legal-position/fraud-plagiarism/`

Instructions

Read the questions and write your answers in a separate file. On Brightspace –together with this Exam– a template document is provided that you should download and use to write your answers.

You can edit the answer document using a document editor of your choice installed on your own computer, e.g. Microsoft Word, as long as you can produce a PDF document to upload.

You do not need to copy the question text, just clearly reference the question you are answering to (e.g. Question 1, Question 2, etc.).

If you are asked to draw or sketch something, you can:

- Use some other tools (on your PC or online) and embed the resulting image into the document.
- Do it by hand on a piece of paper, then scan or take a good picture of it, and embed it into the document.

At the end, create **a single PDF file** of the document with your answers, named: `STUDENTID_SURNAME_exam.pdf` (obviously use your own student number and surname in the filename).

The answer document must contain:

- On the first page: your surname, your name, your student ID, and your study programme, and the fraud/plagiarism awareness statement.
- From the second page: your answers to the questions.

Please check that the text and embedded materials have a proper resolution in the final PDF, i.e. they are clearly visible and readable.

Finally, upload the PDF file before the deadline in Brightspace.

You are allowed to upload your answer PDF document multiple times (so you can improve an earlier version). However, be aware that we will only grade the last submission that you provide.

DEADLINE: You have 3 hours to complete this exam and upload the deliverables in Brightspace.
People who are entitled to extra time have 30 more minutes.

Question 1 (5 points)
True or False?

(a) (1 point) `+`, `-`, / and `*` are called operators
◯ True ◯ False

(b) (1 point) Updating a variable by adding 1 is called an increment, subtracting 1 is called a decrement
◯ True ◯ False

(c) (1 point) *Search* is to find an element in a sequence, by visiting each element from the start one by one. *Lookup* is finding the key/value pair inside a dictionary.
◯ True ◯ False

(d) (1 point) The result of the following Python code is `(1, 2, 3, 4)` being printed to the screen:

```
tup2 = (1, 2, 3)
tup3 = (4)
print(tup2 + tup3)
```

◯ True ◯ False

(e) (1 point) The following Python code will work correctly:

```
dct2 = {10: 1, 20: 2, 30: 3}
print(dct2[1])
```

◯ True ◯ False

Question 2 (10 points)

(a) (6 points) Complete each statement below by filling in the blanks:

1. You use ____________________________ in your Python programs, denoted by the # symbol, to explain in natural language how your program works.

2. In Python, the single equal sign (=) is used for ____________________________,

3. while the double equal sign (==) is used for ____________________________.

   A good recursive function requires three things:

4. First, it must call ____________________________.

5. Second and third, it must have a ____________________________ condition

6. and it must ____ it.

(b) (4 points) Complete each statement below by filling in the blanks. Pick your answer from the following options (note, you are allowed to use an option zero, one or multiple times):

- return
- print
- both print and return
- neither print nor return

1. ___________________________________
   can be used while defining the code for a function.
2. ___________________________________
   can be used outside a function definition.
3. ___________________________________
   can be used to terminate execution inside a function.
4. ___________________________________
   will always show the result on screen.

Question 3 (5 points)

Consider the following two fragments:

Fragment 1:

```
if x == 5:
    x += 1
else:
    x = 8
```

Fragment 2:

```
if x == 5:
    x += 1
if x != 5:
    x = 8
```

Explain in your own words what is the difference between the two fragments.

Question 4 (5 points)

What is the output if you run the following Python code?

(a) (1 point)

```
t = [1, 2, 3, 4, 5]
print(t[1:3])
```

(b) (1 point)

```
print("Geomatics".isalpha())
```

(c) (1 point)

```
i = 0
while i < 5:
     i += 1
     if i == 3:
         continue
     print(i)
```

(d) (1 point)

```
s = "Geomatics is fun!"
print(s.find("is"))
```

(e) (1 point)

```
class Demo:
    def __init__(self, b):
       self.b = b
    def print(self, c):
       print(c + self.b)

a = Demo("Red")
a.print("Blue")
```

Question 5 (5 points)

Explain in your own words what will happen if you leave out the default argument in the following code:

```
default = ['BSc', 'MSc']
for i, e in enumerate(zip(default, ['Geography', 'Geomatics']), start=15):
    print(i, e)
```

In your answer:

1. Indicate what is the default argument
2. Use the term: tuple

Question 6 (5 points)
Explain in your own words why the following code:

```
def mylist_adder(el, lst=[]):
    lst.append(el)
    return lst

l = mylist_adder(1)
s = mylist_adder(2)
t = mylist_adder(3)

print(l)
print(s)
print(t)
```

Does not lead to:

```
[1]
[2]
[3]
```

but to the output:

```
[1, 2, 3]
[1, 2, 3]
[1, 2, 3]
```

Question 7 (5 points)

(a) (1 point) Write a single Python statement to follow the code below that prints the value 'corn' by accessing the right element from the list food.

```
food = ["peas", "carrots", "corn", "potatoes"]
```

(b) (1 point) Assume you have the Python variable friends that refers to a list of strings.

Write Python code to print each element of the list on a separate line.

(c) (1 point) Python list L is a list of lists that could look as shown below. Using a for loop, extract and print the second element of each of the sublists of L, each on its own line.

For example, if L were set as follows:

```
L = [ ["a", "this_one", "b"], ["c", "that_one", "d"] ]
```

your for loop would print:

```
this_one
that_one
```

(d) (2 points) Assume you have the Python variable prices that refers to a list of floating point numbers. Write Python code to print the total of all the elements in prices without using the built-in sum function.

Question 8 (10 points)

Give a Python function `translate` that returns for a series of digits, given as a string, their spoken English counterpart (0 → zero, 1 → one, 2 → two, 3 → three, 4 → four, 5 → five, 6 → six, 7 → seven, 8 → eight, 9 → nine), separated by a user defined delimiter character.

If other (non-numeric) characters are encountered, these should be skipped. If no digits are encountered an empty string should be returned.

An example: If you call the function with the string: `52496` and the dash character (`-`) as delimiter, the return value should be: `five-two-four-nine-six`

Question 9 (10 points)

For recording how many times you play a specific audio track you make a Python program.

Within the program you keep per artist all the tracks which the artist authored and their play count in a (nested) dictionary, as follows:

```
plays = {
    'artist A':
        {'track A': 10, 'track B': 11},
    'artist B':
        {'track C': 4, 'track D': 7},
}
```

Show with Python code how you would:

(a) (2 points) Get the total amount of plays ′Artist A′ with all his tracks got.

(b) (2 points) Increment play count for one of the tracks of an artist.

(c) (2 points) Add a new artist with a new track (you can make up an artist and track name yourself)

(d) (2 points) Remove all information stored for an artist

(e) (2 points) Change a name of a track, but keep its track count

Question 10 (5 points)
Draw a stack diagram for the following Python program:

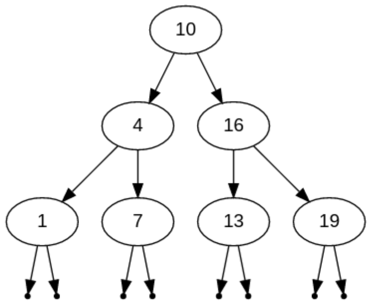
```
def gcd(x, y, depth=1):
    """
    Find the greatest common divisor of x, y
    Pre-condition: x >= y, y >= 0, both x and y are int
    """
    if y != 0:
        rem = x % y
        result = gcd(y, rem, depth + 1)
        return result
    else:
        tmp = x
        return tmp

def main():
    m = 12
    n = 9
    print "Finding gcd({}, {})".format(m, n)
    g = gcd(m, n)
    print("Greatest common divisor of {}, {} = {}".format(m, n, g))

main()
```

Question 11 (10 points)

Given the following class definition (for nodes in a binary tree):

```
class Node:
    def __init__(self, val, left=None, right=None):
         self.value = val
         self.left = left
         self.right = right
```

The following binary tree (the node instances are depicted as circles, the value for each node is inside the circle and the arrows depict their left and right child node references):



can be made as follows:

```
tree = Node(10, Node(4, Node(1), Node(7)), Node(16, Node(13), Node(19)))
```

(note, `tree` refers to the top-most node, with value 10, also called the root node).

Rewrite the following function as a *recursive* function:

```
def size_iter(t):
    """Returns the number of nodes in binary tree T.
    >>> t = Node(10, Node(4, Node(1), Node(7)), Node(16, Node(13), Node(19)))
    >>> size_iter(t)
    7
    """
    count = 0
    unprocessed = [] # unprocessed nodes
    if t is not None:
        unprocessed.append(t)
    while unprocessed: # while unprocessed nodes is not empty
        u = unprocessed.pop() # get an unprocessed node
        count += 1 # increment counter
        if u.left is not None:
            unprocessed.append(u.left) # add left child if it exists
        if u.right is not None:
            unprocessed.append(u.right) # add right child if it exists
    return count
```

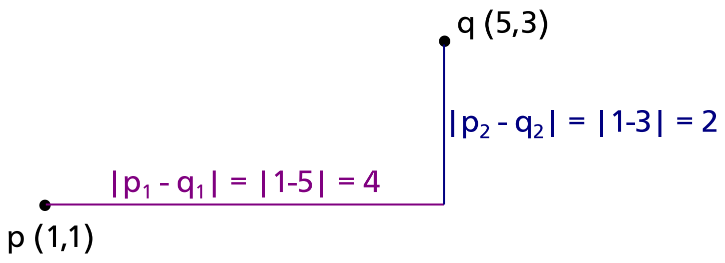
Note, you can assume that the argument t either refers to a Node instance or to None.

Question 12 (10 points)
Make a class definition that conforms to the following description:

(a) (2 points) The class name is `Point2`, and each instance will have 2 properties (attributes) of type float, named `x` and `y`. The constructor will take care of proper casting.

(b) (2 points) The instances should have a method named `cab_distance`, which computes the Manhattan distance between the Point2 itself and another Point2 given.

The Manhattan distance between (p_1, p_2) and (q_1, q_2) can be calculated as: $|p_1 - q_1| + |p_2 - q_2|$ as illustrated in the following Figure:



The `cab_distance` method should follow the type based dispatch pattern allowing to compute the Manhattan distance also to points that are represented as tuples or lists with (at least) 2 numeric values.

(c) (2 points) The class makes available an overloaded method that allows you to get a string representation of the Point2 objects, where Python expects this (e.g. in a print statement). The string produced should have the coordinate values of the instances rounded to 4 digits, separated by comma and a space after the comma.

With the class definition available, show code that:

(d) (2 points) creates two instances, named `p` and `q`. Point `p` is located at $(1, 1)$ and point `q` at $(5, 6)$.

(e) (2 points) prints correctly the Manhattan distance between these two instances.

Question 13 (15 points)

During these challenging COVID-19 times, you decide to start your own web shop and deliver toilet paper rolls. For delivery you cannot carry more than a number of toilet paper rolls at the same time on your bike. You write a Python program that helps you to create a listing of your orders for delivery. However, you are not exactly sure how many rolls you can carry at the same time. Therefore you like the program to be flexible making the amount of paper rolls you can deliver at the same time configurable. You write all addresses and how many rolls were ordered per address as tuples (of a string and an integer) in a list (see below for sample input). Give a Python program, that prints a listing of how you should deliver all toilet paper rolls in batches, making every batch not larger than the amount that you set that you can carry (for which you can assume it will be $>= 1$).

Your program should print per batch (identified by a unique number) the total number of items to be delivered, the delivery addresses and the item count to be delivered (see below for sample output). If, by adding a order its count to a batch goes over the maximum amount you can carry, this should start a new delivery batch with the remaining items of the order. The same logic should also handle when someone ordered more than the maximum amount (the order for such a person will be split over multiple batches).

Note: there is *no* need to optimize – i.e. compute the absolute minimum number of batches you could have for delivery – make sure all batches have not more than the maximum toilet paper rolls you can carry.

Sample input:

```
[("Address 1", 4),
 ("Address 2", 15),
 ("Address 3", 3),
 ("Address 4", 8)]
```

with maximum amount set to 7, should output:

```
# Toiletpaper delivery

## Batch 1 - 7 items
- Address 1: 4
- Address 2: 3

## Batch 2 - 7 items
- Address 2: 7

## Batch 3 - 7 items
- Address 2: 5
- Address 3: 2

## Batch 4 - 7 items
- Address 3: 1
- Address 4: 6

## Batch 5 - 2 items
- Address 4: 2
```

End of Exam