Advanced Programming

XXX

MOCK Exam 1 SOLUTIONS

XXXX,
Duration of the exam is 90 minutes

The exam was made by Lorenzo Galeotti & Breanndán Ó Nualláin and peer-reviewed by Lorenzo Galeotti & Breanndán Ó Nualláin

Student name_____

Teacher name_____

Amsterdam University College
Advanced Programming

Instruction:

- You are allowed to use a blue or black pen, but nothing else.
- Please write your name on the front page before you answer the questions.
- Please do not remove the staple from the exam and keep all sheets connected!
- Write your answers on this exam, in the empty space after the question.
- Please switch off any electronic devices and put them in your bag. Your bag needs to be closed and placed at the front wall of the classroom. During the exam you may have only a pen and a water bottle at your desk.
- Any violation of AUC's rules on fraud may lead to sanctions, ultimately to the exclusion of all examinations for one year (AS&P appendix 2, Regulations governing fraud and plagiarism).
- The number of points per question is specified in the table below.

The result of this test has a weight of 28% in your final grade. The points of the individual questions are:

Part I: $2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 = 20$;
Part II: $5 + 5 + 15 + 5 = 35$;
Part III: $5 + 10 + 30 = 45$.

Your exam grade will be: (points received)/10

Part I: Theoretical Questions

Question 1 (2pts.) How do you display the contents of a file called `example.txt` on the terminal?

Answer:

`less example.txt` or `cat example.txt`

Question 2 (2pts.) How do you redirect the standard output of the `date` command to a file named `output.txt`, overwriting the file if it exists?

Answer:

```
data > output.txt
```

Question 3 (2pts.) What command would you use to count the number of lines in a file called `notes.txt`?

Answer:

```
wc -l notes.txt
```

Question 4 (2pts.) What is the command to find out your current working directory?

Answer:

```
pwd
```

Question 5 (2pts.) Explain the difference between the following commands:

```
echo "hello" | wc
echo "hello" > wc
```

Answer:

The first command will produce the number of lines, words, and characters in the string "hello" (1 1 6 note that this includes the new line character). The second line will create a file called wc and print the string "hello" on the first line of the file wc.

Question 6 (2pts.) What command initializes a new Git repository in the current directory?

Answer:

```
git init
```

Question 7 (2pts.) How do you commit staged changes with a message "Initial commit"?

Answer:

```
git commit -m "Initial commit"
```

Question 8 (2pts.) What command lists all branches in your Git repository?

Answer:

```
git branch
```

Question 9 (2pts.) What does the following command do?

```
git clone https://github.com/user/repo.git
```

Answer:

Clones a repository from the URL to your local machine.

Question 10 (2pts.) Briefly explain how *Git* branching works and how can be used to collaborate on big projects.

Answer:

When working on a repository, branching allows us to create parallel lines of work. Typically, whenever a team wants to add a new feature to their project, a new branch is made. The team will then work on this new branch until the new feature is fully developed and ready to be integrated into the main project. This is when the new branch will merge with the main branch.

Part II: Scoping, Mutability, & Recursion

Question 11 (5pts.) What will the following Python code snippet print out? Explain why.

```
x = [2, 3, 4]
y = x

print(x, y)
```

Answer:

The program will print $[2, 3, 4]$ twice. This is because x and y will refer to the same object, the list.

Question 12 (5pts.) What will the following Python code snippet print out? Explain why.

```
x = [2, 3, 4]
y = x
x = [7, 8, 9]

print(x, y)
```

The program will print $[7, 8, 9]$ and $[2, 3, 4]$. Indeed, after the assignment $y = x$ we have that both x and y contain a reference to $[2, 3, 4]$, then after the assignment $x = [7, 8, 9]$ the variable x will contain a reference to $[7, 8, 9]$.

Question 13 (15pts.) Consider the following two functions:

```
def f(n,m):
    if n == 0:
        return 0
    else:
        return m+f(n-1,m)
```

```
def g(n,m,p):
    if n == 0:
        return p
    else:
        return g(n-1,m,p+m)
```

1. Assume that m and n are natural numbers. What are the values computed by $f(n, m)$ and $g(n, m, 0)$?
2. Which one of the two function is *tail-recursive*? Explain your answer.

Answer:

1. Both functions are computing the product of m and n.
2. A function call of a function g is called **tail call** if it is of the type `return g(...)`, i.e., the caller returns the value returned by g without further calculation. According to this definition function g is tail-recursive and function f is not.

Question 14 (5pts.) What will the following Python code snippet print out? Explain why.

```
x = 42


def f():
    x = 0
    y = x
    return [x, y]


print(f(), x)
```

Answer:

The code will print $[0, 0]$ 42. This is because name x within the function f refers to a new variable that shadows the global variable x. Therefore f just returns the list $[0, 0]$ and the global x is unchanged during the execution of the entire program.

Question 15 (5pts.) What will the following Python code snippet print out? Explain why.

```
def make_counter():
    n = 0

    def counter():
        nonlocal n
        n += 1
        return n

    return counter


c = make_counter()
for i in range(4):
    print(c())
```

Answer:

The program will print 1 2 3 4. Indeed, c will be an instance of the function counter which every times that is called increments n by 1 and returns the new value of n. Given that we call c four times in the for loop and print the result every time, the function will print the integers from 1 to 4.

Part III: Programming

Question 16 (5pts.) Write a Python function which takes a file name and returns a list containing the number of characters of each line of the file.

Answer:

```
def print_char(filename):
    with open(filename) as file:
        return [len(line.strip('\n')) for line in file]
```

Question 17 (10pts.) Create a generator function in Python that allows you to iterate over the Fibonacci sequence $(0, 1, 1, 2, 3, 5, \ldots)$. Show how this function can be used to print the first 10 Fibonacci numbers.

Answer:

```
def gen_fib():
    n = 0
    m = 1
    yield n
    yield m
    while True:
        n,m=m,n+m
        yield m

g=gen_fib()
for n in range(10):
    print(next(g))
```

Question 18 (30pts.) We can define binary trees of integers in Lisp as follows

```
(defstruct node value left right)

(defstruct node
  (value  nil :type fixnum)
  (left   nil :type bt)
  (right  nil :type bt))
```

Recall that a binary search tree is a binary tree with the following property:

> For every node n of the tree the values of the nodes in its left subtree are smaller or equal to the value of n and the values of the nodes in the right subtree of n are strictly larger than the value of n.

In the following exercises you can assume that our trees contain only positive integers.

1. Write a Lisp function that given a binary tree t and two values n and m substitutes all the occurrences of n in t with m.
2. Write a Lisp function that given a binary tree checks whether it is a binary search tree or not.
3. Write a Lisp function that takes a binary search tree and a value v and returns the smallest value in the tree which is strictly larger than v. If all the values in the tree are smaller than or equal to v your function should return -1.

Answer:

```
(defun subst(bt n m)
  (cond ((null bt) nil)
        ((node-p bt) (let ((v (node-value bt))
                           (l (node-left bt))
                           (r (node-right bt)))
                       (node (if (= v n)
                                 m
                                 v)
                             (subst l n m)
                             (subst r n m))))))

(defun bst-p (bt)
  (cond ((null bt) t)
        ((node-p bt) (let ((v (node-value bt))
                           (l (node-left bt))
                           (r (node-right bt)))
                       (and (or (null l) (<= (node-value l) v))
                            (or (null r) (<= v (node-value r)))
                            (bst-p l)
                            (bst-p r))))))
```