TU DELFT – FACULTY OF AEROSPACE ENGINEERING

Course	:	Programming and Scientific Computing in Python (AE1205)
Date	:	July 7th, 2016 from 13:30 until 16:30 hr
Lecturer	:	prof. dr. ir. Jacco Hoekstra, dr. ir. René van Paassen
Remarks	:	Write your name, initials and student number on your work. Answer all questions in English and mark all pages with your name and the date of the exam.

Instructions

NOTE: For Part I, II and III of the exam, you must use a (white) answer sheet. Computer use is not allowed for these parts. Do not switch on your computer/monitor, and work only with pen, paper and calculator.

After you completed parts I, II and III, raise your hand. You then hand in the answer sheet to an invigilator, and will receive a yellow(!) answer sheet for Part IV of the exam.

For Part IV you may use the computer. You can use the IDLE or Spyder development environments, and interactive python help through the interpreter (e.g. use F1 Help). You may also use the Python documentation installed on the computers, to access this from Spyder, select "Help" and "Installed Python Modules" for the Matplotlib, Numpy or Scipy documentation.

Warning: Computer use is only allowed when working with the yellow answer sheet. Any other computer use must be reported as exam fraud.

Allowed Items

Parts I, II and III:	Reader AE1205 Scrap paper, ruler, protractor (=geodriehoek) Normal (non-graphical, non programmable) calculator.
Part IV:	Reader AE1205 Scrap paper (use it!), ruler, protractor (=geodriehoek) Computer, Python, Python(x,y) documentation (local only)

A total of 100 points can be earned at the exam while you need only 90 for the maximum score. This means you do not need to solve all 4 problems of part IV for the maximum score, so use your time wisely. The list below shows the number of points that can be earned for each question. The final grade will be 1 + npoints/10, and rounded off as per TU Delft policy.

Questions 1 to 8 :	3 points per question	(24 in total for part I and part II)
Questions 9 :	24 points	(3 points per answer for part III)
Questions 10	10 points	
Questions 11 :	16 points	
Questions 12 :	10 points	
Questions 13 :	16 points	

Part I: What will this print? (5 x 3 = 15 points)

(We only need to see that you understand the logic, exact brackets, commas, quotes and newleines are irrelevant for the grading.)

1. What will the following program print?

```
a = range(0,20,3) + range(0,20,4)

print a[::-2]
```

2. What will the following program print?

```
A = [[1,2,3],
     [4,5,6],
     [7,8,9]]

tritotal = 0

for i in range(3):
    for j in range(i+1):
        tritotal = tritotal + A[i][j]

print tritotal
```

3. What will the following program print?

```
i = 2
while i<12:

    swp = True
    for k in range(2,i):
        if i%k==0:
            swp = False
    if swp:
        print i

    i = i + 1
```

4. What will the following program print?

```
from numpy import *

def f(x):
    return x*x

a    = linspace(0,50,51)
cond = f(a)<30

print sum(cond*a)
```

5. What will the following program print?

```
from numpy import *

g  = 10.0 # [m/s2]
h0 = 10.0 # [m]

t = arange(0,10.1,0.1)     # [s]
y = h0 - 0.5*g*t*t         # [m]

ft = t[y>=0.0]    #[s]

print "time = ",max(ft)," s"
```

Part II: Debug the program (3 x 3 = 9 points)

6. A beginning programmer tries to make a program to solve 2nd order polynomials. The following test program does not run but shows a message "**There is an error in your program: invalid syntax**" It highlights **the return statement**. How should he fix this bug?

```
from math import *

def solve(a,b,c):
    D = b*b - 4.0*a*c
    sol = []
    if D > 0.0:
        x1 = (-b - sqrt(D))/(2.0*a)
        x2 = (-b + sqrt(D))/(2.0*a)
        sol = [x1,x2]
    else:
        sol = [-b/(2.0*a)]
    return sol

a = 1
b = 2
c = -3.0

print "Solutions for",a,",",b,"and",c,"are:",solve(a,b,c)
```

7. The junior Python programmer keeps on struggling with the relatively simple program from the previous problem and tries to run the following version. What will happen now?

```
from math import *

a = 1
b = 2
c = -3.0

def solve(a,b,c):
    D = b*b - 4.0*a*c
    if D > 0.0:
        x1 = (-b - sqrt(D))/(2.0*a)
        x2 = (-b + sqrt(D))/(2.0*a)
        return [x1,x2]
    else:
        return []
    return

print "D = ",D
print "Solutions for",a,",",b,"and",c,"are:",solve(a,b,c)
```

A. The program will work flawlessly and solve the equation $x^2 + 2x - 3 = 0$
B. The program will not run because of a syntax error in the function definition
[✓ C.] The program will not run because of an import error
D. The program will run, but stop due to the wrong position of defining the function
E. The program will run, but not print the solutions as no results are returned
F. The program will stop due to a Name Error, as the variable D is not defined
G. The program will run, but stop with a Type Error as it tries to evaluate a function with multiple, different returns

8. A programmer tries to make an interpolation function. Still, the test program keeps printing the result 22.0 for the test values given in the program below.

   Which line of the **function definition** should be changed to make sure the program prints the given correct result of the test case?

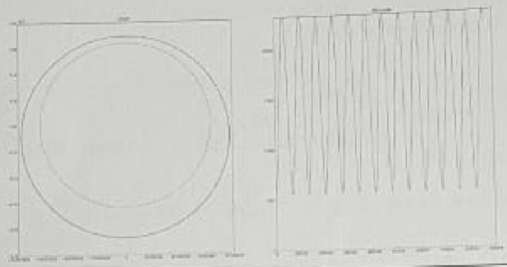
```
def interpol(x,a,b,fa,fb):
    factor = float((x-a)/(b-a))
    y      = (1.0-factor)*float(fa) + factor*float(fb)
    return float(y)

# Test values: should give 28.2857142857
a  =  3
b  = 10
fa = 22
fb = 44

print interpol(5,a,b,fa,fb)
```

   A. The line: `def interpol(x,a,b,fa,fb):`
   B. The line: `factor = float((x-a)/(b-a))`
   C. The line: `y      = (1.0-factor)*float(fa) + factor*float(fb)`
   D. The line: `return float(y)`
   E. The indentation of the two lines in between **def** and **return**
   F. This problem cannot be solved inside the function but can only be solved by adapting the main program

(exam continues on next page)

Part III: Complete the program (8 x 3 = 24 points)

9. The program below should simulate the orbital manoeuvres of a satellite. The goal is to plot two plots next to each other. Fill the gaps in the program underneath the figure.



```
from numpy import *
from matplotlib import pyplot as plt

Me = 5.972e24   # [kg] mass earth
Re = 6.371e6    # [m] radius earth
G  = 6.674e-11 #[m3/kg s] gravitational constant

# Initial condition
msat   = 2.2     # [kg] Delfi C3 mass
h      = 5.62e5 # [m]  initial altitude
v0     = 8e3     # [m/s] initial radial velocity

# Cartesian start position & velocity
x = 0.
y = Re+h
vx = -v0
vy = 0.0

# Make circle for spherical earth with steps of 1 degree
phi     = ________(a)________  # [deg]
phirad = radians(phi)          # [rad]
xcir    = Re*sin(phirad)       # [m]
ycir    = Re*cos(phirad)       # [m]

# Start the clock
t  = 0.0             # [s] time
dt = 0.1             # [s] time step
day = 24.*3600. # simulate one day

# Create empty tables
xtab,ytab,htab,ttab = [],[],[],[]

#    **** continued on next page ****
# Run one day or until crash into spherical earth
while t<=day _____(b)_____ :

    # Calculate force of gravity
    r  = sqrt(x*x+y*y)
    Fg = G*msat*Me/(r*r)

    # Calculate x,y components
    theta = _____(c)_____

    Fgx = -Fg*cos(theta)
    Fgy = _____(d)_____

    # Numerical integration
    ax = ____(e)_____
    ay = ____(f)_____

    vx = vx + ax*dt
    vy = vy + ay*dt

    x = x + vx*dt
    y = y + vy*dt

    h = r - Re

    t = t + dt

    # Add to tables
    xtab.append(x)
    ytab.append(y)
    htab.append(h/1e3)
    ttab.append(t)


# Show orbits
plt.subplot(121)
plt.title("Orbit")
plt.plot(xtab,ytab) # orbit
plt.plot(____(g)____) # red line indicating spherical earth

# Show altitude (assuming spherical earth)
plt.subplot(122)
plt.title("Altitude")
altmax = max(htab)
plt._____(h)_____    #range of y-axes: from 0 to max alt
plt.plot(ttab,htab)

plt.show()
```

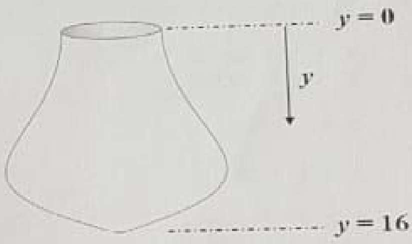
Part IV Programming Python (10+16+10+16 = 52 points)

Submit your solutions answer + by saving your files! Read here how.
On the **answer form** a one or two sentence **description** of your method and the **final answer** that you have found..

Also save your scripts on the **H:** drive on your computer. Make a **folder** which is just your student number (e.g. 4345678) . Then use the following naming convention for your programs, start all program files names with a 'p' followed by the problem number. So p10-wine.py is a good name, p11-trebles.py etc. You're free to use any name you want after the 'p10', 'p1'' etc. but it has to start with these three characters So we can find your program automatically as : **H:/4345678/p10*.py, H:/4345678/p11*.py**, etc..

As always, adding comments, a header with student name+number and clear variable naming is appreciated.

10. **Decanting wine (10 points)**
The vase on the right can be described as a cylinder with a varying radius. Using a y-coordinate which starts at the top and then runs all the way down, the shape can be described by the radius in cm using the y-coordinate in cm:



$$r(y) = 2 + y\sin\left(\sinh\left(\frac{1}{8}y\right)\right)$$

$$h = 16 - y$$

(with r, h and y in centimetres)
(the function sinh( ) or hyperbolic sine, can be found in the math module)

A wine decanter is used to let the wine 'breathe' a bit before drinking it and to avoid serving glasses with the sediment of an old wine. A student, who somehow hasn't got a crystal wine decanter, is inventive and decides to use this simple, cleaned vase as to decant the wine.

How high will the surface of the wine be, measured from the bottom, when a bottle of wine of 0.750 litres (so 750 cm^3) is poured in this vase? Calculate the volume by summing the small volumes of the flat cylinders of with a height of Δy given by the volume of this disc: $\boxed{\Delta V = \pi r^2 \Delta y}$ Starting from the bottom of the phase.

To check your program: a volume of 0.25 litres will fill it up to a height of h = 2.736 cm (rounded to three decimals).

Give your answer h in cm for a complete bottle of 0.750 dm3, rounded off to 1 digit behind the decimal point.

11. Trebles of 5 digits (16 points)

A treble means a number which is a multiple of 3. In this problem we are looking for 5 digit numbers of which the original number as well as their treble use the same 5 digits (in a different order obviously).

An example is **28575**: as 28575 x 3 = 85725, so both the original single as well as the treble conmsist of exactly the same digits: one 2, one 7, one 8 and two 5's.

The question now is:
List all the original, single five digit numbers which exhibit this property of having the same digits as their treble. **Find the complete list of 13 of these five-digit numbers (and their sum).**

Make a program, call it p13-trebles.py, which prints a number of lines numbered with 1,2,3 etc. and then prints the original 5-digit number and to let you check it, it also shows its treble with the same digits on the same lines. It concludes by printing how many it has found and what the total is. (Hint: The amount of numbers it should find is in the range between 5 and 20.)

As an example solution: if do this for four digit numbers, you should find only 2 of such numbers:

```
   1.   1035 (because x 3 = 3105)
   2.   2475 (because x 3 = 7425)
The sum these numbers is 3510    (1035+2475)
```

On the answer form, describe briefly in one or two sentences your method, then list 13 found five-digit numbers (only the singles, but not their trebles) and their sum in the answer box. So for the example above, directly after your brief description of the method, you would write: 1035 + 2475 = 3510

12. Sorting Monty in Python (10 points)

Sort the characters of the following conversation in the following way:

- First collect all vowels (Dutch klinkers): AEIOU, sort them alphabetically
- Then all consonants (Dutch: medeklinkers), also sort them alphabetically
- Put these characters in a list or one long string.
- Leave all spaces and symbols out. Make no distinction in upper case or lower case

If you're done, your complete list or string, with first all sorted vowels and then all sorted consonants, should be 84 characters long (check the length of your list/string), it should start with more than 10 a's and end with 3 y's. Check: at the 47th position (index 46) you should find an "L".

Write down the **six characters** at position **45, 50, 17, 38, 42 and 73 when we start counting with 1 for the first character** (one of the "A"'s), 2 for the second character (also an A), etc. In other words: list the characters at indices **44, 49, 16, 37, 41, 72**

```
ARTHUR:   Now stand aside, worthy adversary.
BLACK KNIGHT:   'T is but a scratch.
ARTHUR:   A scratch?  Your arm's off!
```

13. A classic: the travelling salesman problem (16 points)

The travelling salesman problem is a classical problem which cannot be solved mathematically but only by trying, so with brute force. With only a limited number of cities the search space is not too large for such a brute force approach. And with Python we can use our PC as our mathematical slave.

In our case, a Jokes & Novelties salesman, who lives in Delft, needs to visit the **list of Dutch cities in the table below**. He wants to know three options for the shortest routes, not counting a reversed route as a separate route. Can you help him?

All routes should start and stop in Delft. He should visit **all other cities just once.** The route length should be measured starting in Delft and including the final leg back to Delft using the distances between the cities in the route.

Copy the table with city coordinates in your program and check the totals of x and y to see if you've copied them without mistakes.

Use the example route generating program on the lower right as inspiration and/or starting point for your own program. In this program idxlist is a list of possible city indices for a route[0,1,2]. Then see how a for-loop with itertools.permutations( ) is used to shuffle this list of indices in such a way that irte will be a copy of this list but then in all possible orders (permutations). This shuffled index list irte is then used to print the route each time in the for-loop. The example output below the program..

Make a program to find the three shortest routes. On the answer form list **the three shortest routes and also give also the length in km of the these routes (you can use the first character of each city as short-hand).** The same route reversed should not appear in the list and you can use the straight line distance between the cities.

	x [km]	y [km]
Amsterdam	37	39
Delft	0	0
Nijmegen	101	-22
Zwolle	118	56
Groningen	151	134
Leeuwarden	99	132
Rotterdam	8	-10
Utrecht	52	9
Enschede	174	23
Breda	28	-49

Check totals: x : 768 y : 312

```
import itertools

cities = ["New York","Boston","San Francisco"]
idxlist = range(3)

for irte in itertools.permutations(idxlist):
    print "Home -",

    for icity in irte:
        print cities[icity],"-",

    print "Home"
```

```
>>>
Home - New York - Boston - San Francisco - Home
Home - New York - San Francisco - Boston - Home
Home - Boston - New York - San Francisco - Home
Home - Boston - San Francisco - New York - Home
Home - San Francisco - New York - Boston - Home
Home - San Francisco - Boston - New York - Home
>>>
```