Front page

UNIVERSITY OF OSLO
Faculty of mathematics and natural sciences

Examination in: IN1900/INF1100 — Introduction to programming for scientific applications
Day of examination: December 18th 2017
Examination hours: 9.00-13.00
Attachments: 1 (ForwardEuler.pdf)
Permitted aids: None

- Read through the complete exercise set before you start solving the individual exercises. If you miss information in an exercise, you can provide your own reasonable assumptions as long as you explain them in detail.
- All code in the question texts is written in Python 3. You can write your answers in either Python 2 or Python 3, but you should avoid using a mix.
- Most of the exercises result in short code where there is little need for comments, unless you do something complicated or non-standard. In that case, comments should convey the idea behind the program constructions such that it becomes easy to evaluate the solution.
- Many exercises ask you to “write a function”. A main program calling the function is then not required, unless it is explicitly stated. You may, in these types of exercises, also assume that necessary modules are already imported outside the function. On the other hand, if you are asked to write a complete program, explicit import of modules must be a part of the solution.
- The maximum possible score on this exam is 84 points. There are 24 questions in total, and the number of points is specified for each individual exercise.

1.1 What is printed?

What is printed when the following code is run?

```
s = -2
for k in range(2, 5, 2):
    s += 2
print(s)
```

Select an alternative:

- 0
- -2
- 2 ✔
- 4

Maximum marks: 2

1.2 What is printed?

What is printed in the terminal window when the following code is run?

```
a = [8, 9, 10, 11]
b = a[1:-1]
a[1] = 10
print(b[0])
```

Select an alternative:

- 8
- 11
- 10
- 9 ✔

Maximum marks: 2

1.3 What is printed?

What is printed in the terminal window when the following code is run? The argument **end = ' '** in the **print** calls is Python 3 syntax, and replaces the usual line shift after a **print** call with a space character.

```
for i in range(2, 5):
    print(i,end = ' ')
    for j in range(i-1, i+1):
        if i != j:
            print(j, end = ' ')
```

Select an alternative:

- 2 1 3 2 4 3 ✔
- 2 2 1 1
- 2 1 2 1
- 1 2 2 3
- 2 1 3 2

Maximum marks: 2

1.4 What is printed?

What is printed when the following code is run? Recall that the function **enumerate(a)**, if a is a string of length n, will create a list of tuples [(0,a[0]), (1,a[1]), ..., (n-1,a[n-1])].

```
def freq_lists(dna_list):
    n = len(dna_list[0])
    A = [0]*n
    T = [0]*n
    G = [0]*n
    C = [0]*n
    for dna in dna_list:
        for index, base in enumerate(dna):
            if base == 'A':
                A[index] += 1
            elif base == 'C':
                C[index] += 1
            elif base == 'G':
                G[index] += 1
            elif base == 'T':
                T[index] += 1
    return A, C, G, T
dna_list = ['GGTAG', 'GGTAC', 'GGTGC']
A, C, G, T = freq_lists(dna_list)
print(A)
```

Select an alternative:

- [0,0,0,2,0] ✔
- [2,0,0,0]
- [0,0,0,1,0]
- [0,0,0,'A',0]
- ['C','G','T','A','G']

Maximum marks: 4

1.5 What is printed?

What is printed in the terminal window when the following code is run?

```
import numpy as np

def fibonacci(N=3):
    x = np.zeros(N+1, int)
    x[0] = 1
    x[1] = 1
    for n in range(2, N+1):
        x[n] = x[n-1] + x[n-2]
    return n, x[n]

print(fibonacci(N=2))
```

Select an alternative:

- (2,2) ✔
- [0,1,1,2,3]
- [0,1,1,2]
- (1,1,2,3)

Maximum marks: 2

1.6 What is printed?

What is printed in the terminal window when the following code is run? If you believe nothing is printed, you should explicitly write this, i.e. "Nothing is printed".

```
def primeTable(k):
```

```
    n = 5
    primelist = [2,3,5,7,11]
    print(primelist)

    diff = primelist
    for i in range(1, k):
        for j in range(n-i):
            diff[j] = abs(diff[j+1] - diff[j])
        print(diff[:n-i])

primeTable(2)
```

Fill in your answer here

Format | Σ

Words: 0

Maximum marks: 4

1.7 What is printed?

What is printed in the terminal window when the following code is run?

```
dna_list = ['GGTAG', 'GGTAC', 'GGTGC']
print(dna_list[len(dna_list)])
```

Select an alternative:

- An error message ✔
- GGTGC
- ['GGTAG', 'GGTAC', 'GGTGC']
- 'C'

Maximum marks: 2

1.8 What is printed?

What is printed in the terminal window when the following code is run?

```
import numpy as np
a = np.linspace(0, 5, 1)
a.append(6.0)
print(a)
```

Select an alternative:

- An error message ✔
- [0. 1. 2. 3. 4. 5. 6.]
- [0. 1. 2. 3. 4. 5.]
- [6. 0. 1. 2. 3. 4. 5.]

Maximum marks: 2

1.9 What is correct?

One of the following statements is correct. Which one?

Select an alternative:

- A test function should always include a return statement.
- A test function returns 0 if the test passes.
- A test function should always have at least one argument.
- A test function can have multiple assert statements. ✔
- If a test function runs silently (without an error message), the function being tested is incorrect.

Maximum marks: 2

1.10 What is correct?

One of the following statements is correct. Which one?

Select an alternative:

- Adding two Numpy arrays of length n will result in an array of length 2n.
- The call numpy.sin(2) will give an error message, since 2 is not an array.
- Vectorization means to avoid explicit for-loops in the code. ✔
- Numpy arrays can only be used for storing numbers.

Maximum marks: 2

1.11 Lists

Which of the following expressions does **not** result in a list of length 5?

Select an alternative:

- [0]*5
- list(range(2))+list(range(3))
- [e**2 for e in range(1,5)] ✔
- [2,3]+[0,2,3]

Maximum marks: 2

1.12 What is printed?

What is printed in the terminal window when the following code is run?

```
class Hello:
    def __call__(self, arg):
        return "Hello, %s" % arg

    def __str__(self):
        return "Hello, world!"

def main():
    a = Hello()
    print(a('students'))
    print(a)

main()
```

Fill in your answer here

Format | Σ

Words: 0

Maximum marks: 2

1.13 What is printed?

What is printed in the terminal window when the following code is run? If you believe nothing is printed, you should explicitly write this, i.e. "Nothing is printed".

```
def harmonic(M):
    s = 0
    for k in range(1, M+1):
        s += 1.0/k
    return s

def test_harmonic():
    M = 3
    expected = 1 + 1.0/2 + 1.0/3
    computed = harmonic(M)
    success = abs(expected-computed) < 1E-14
    message = 'Error detected'
    assert success, message

test_harmonic()
```

Fill in your answer here

Format | Σ

Words: 0

Maximum marks: 2

1.14 What is printed?

What is printed in the terminal window when the following code is run?

```
def myfunc(a, b):
    c = a.copy()
    for k in b:
        if k in c:
            c[k] += b[k]
        else:
            c[k] = b[k]
    return c

print(myfunc({1:-1, 3:1}, {1:3, 2:2}))
```

Fill in your answer here

Format | Σ

Words: 0

Maximum marks: 2

1.15 What is printed?

What is printed in the terminal window when the following code is run?

```
def poly_diff(p):
    dp = {}
    for j in p:
        if j != 0:
            dp[j-1] = j*p[j]
    return dp
p = {0:1,1:1,2:1}
```

print(poly_diff(p))

Select an alternative:

- {0: 1, 1: 2} ✔
- {0:2,1:2,2:1}
- {0:1,1:2,2:0}
- {0:1,1:1,2:1}
- [1,2]

Maximum marks: 2

2.1 Heaviside function

The Heaviside function is defined as:

$$H(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Write a Python function **heaviside(x)**, which implements this function.

Fill in your answer here

```
1
```

Maximum marks: 3

2.2 Test function

Write a test function for the function **heaviside(x)** from the previous exercise. Choose x values -1.0 and 1.0, and compare the function result with the expected values 0.0 and 1.0.

Fill in your answer here

```
1
```

Maximum marks: 4

2.3 Numerical differentiation

The derivative of a mathematical function *f(x)* can be estimated with a finite difference approximation:

$$f'(x) \approx \frac{f(x+h)-f(x)}{h}$$

for some small number *h.*

Implement a Python function **diff(f,x,h)**, which applies this formula to estimate the derivative of a function *f* in a point *x*. The function shall return the function value *f(x)* and the estimated derivative. The argument f can be any mathematical function written in Python, that has a single input argument and a single return value.

Fill in your answer here

```
1
```

Maximum marks: 5

2.4 Tabulated output

Write a Python program that evaluates the accuracy of the function **diff(f,x,h)** for estimating the derivate of *sin(x)*. The program shall choose $x = \pi/4, h = 0.5^i$ for i=1,2,3,4, and compute the error, which is the absolute value of the difference between the numerical derivative and the analytical value $-cos(\pi/4)$. Finally, the program shall print out x, h, and the error in a nicely formatted table, as listed below. You can assume that the function **diff(f,x,h)** is written in the same file as your program, so you don't need to repeat or import this function. Include other necessary imports. The output from the program shall look like this:

```
0.785  0.500  0.202
0.785  0.250  0.095
0.785  0.125  0.046
0.785  0.062  0.023
```

Fill in your answer here

```
1
```

Maximum marks: 5

2.5 Prime numbers

A prime number is an integer k > 1 that is only divisible by itself and 1. In other words, to determine if k is a prime, we only have to show that k is not divisible by any of the integers 2, 3, ..., k-1. To test this in Python, note that k is divisible by j if and only if the Python expression k % j == 0 is **True** (for example, 12 is divisible by 6 since 12 % 6 == 0 is **True**). Note: there are much more efficient ways of determining if k is a prime number or not, but we do not consider this here.

Write a Python function **is_prime(k)** that returns the value True if k is a prime number, and returns False if k is not a prime number. You can assume that k is a positive integer.

Fill in your answer here

```
1
```

Maximum marks: 5

3.1 Forward Euler function

The following function implements the Forward Euler method for solving an ordinary differential equation (ODE):

```
def ForwardEuler(f, U0, T, n):
    """Solve u'=f(u,t), u(0)=U0, with n steps until t=T."""
    t = np.zeros(n+1)
    u = np.zeros(n+1)  # u[k] is the solution at time t[k]
    u[0] = U0
    t[0] = 0
    dt = T/float(n)
    for k in range(n):
        t[k+1] = t[k] + dt
        u[k+1] = u[k] + dt*f(u[k], t[k])
    return u, t
```

We want to use this function to solve the ODE
$u'(t) = -u, u(0) = 1$
Which of the following function calls is correct?

Select an alternative:

- u, t = ForwardEuler(-u, U0=1, T=4, n=20)
- u, t = ForwardEuler(lambda u, t: -u, U0=1, T=4, n=20) ✔
- u, t = ForwardEuler(lambda u: -u, U0=1, T=4, n=20)
- u, t = ForwardEuler(f=-u, U0=1, T=4, n=20)

Maximum marks: 3

3.2 Forward Euler class

The attached PDF file contains an incomplete class implementation of the ForwardEuler method. An example of using the class is as follows:

```
def f(u,t):
    return -u

solver = ForwardEuler(f,1.0,10,100)
u,t = solver.solve()
```

The implementations of the functions **solve** an **advance** are missing. Write the code for these functions here. You can write the functions as they would appear inside the class, without repeating the actual class definition or the **__init__** function.

Fill in your answer here

```
1
```

Maximum marks: 10

3.3 Heun's method

Heun's method is a second order method for solving an ODE system *u'(t) = f(u,t)*. The method uses the following formula for advancing the solution from step k to step k+1:

$$u_* = u_k + \Delta t f(u_k, t_k)$$

$$u_{k+1} = u_k + \frac{\Delta t}{2} f(u_k, t_k) + \frac{\Delta t}{2} f(u_*, t_{k+1})$$

Implement Heun's method as a class **Heun**, which is a subclass of the **ForwardEuler** class from the previous question. Inherit as much functionality as possible from the base class, and implement only the necessary parts. You can assume that the **Heun** class is written in the same file as the class **ForwardEuler**, so you do not need to repeat or import the **ForwardEuler** class.

Fill in your answer here

```
1
```

Maximum marks: 5

3.4 Modeling with ODEs

The following ODE system describes interactions between two species; a prey (x) and a predator (y):

$$x'(t) = rx - axy$$
$$y'(t) = -my + bxy$$

The parameters r, a, m, and b are all constants. Write a Python function **predator_prey(x0,y0,r,a,m,b,T,n)**, which uses the **Heun** class from the previous question to solve this system from *t = 0* to *t = T*. The arguments to the function are the initial values (x0,y0), the model parameters (r,a,m,b), the end time (T), and the number of time steps (n). The function shall return three arrays containing the time points for the solution and the two solutions x and y. You can assume that you write the function in the same file as the **Heun** class, so you don't have to repeat or import this class.

Also write the code for calling the function to solve the system for r = m = 1, a = 0.3, b = 0.2, x0=1, y0 = 1, T = 20 and n= 100, and for plotting the two solutions x and y in the same plotting window. Include necessary imports.

Fill in your answer here

```
1
```

Maximum marks: 10

```
import numpy as np
class ForwardEuler:
    """
    Class attributes:
    t: array of time values
    u: array of solution values (at time points t)
    k: step number of the most recently computed
    solution
    f: callable object implementing f(u, t)
    U0: initial condition (scalar or array)
    """

    def __init__(self, f,U0, T, n):
        if not callable(f):
            raise TypeError('f is not a function')
        self.f = lambda u, t: np.asarray(f(u, t))
        self.t = np.linspace(0,T,n+1)
        self.k = 0
        if isinstance(U0, (float,int)): # scalar ODE
            self.neq = 1
            self.u = np.zeros(n+1)
        else: # system of ODEs
            U0 = np.asarray(U0)
            self.neq = U0.size
            self.u = np.zeros((n+1,self.neq))
        self.U0 = U0

    def solve(self):
        """Solve the ODE from time 0 to T.
        Store solution in self.u.
        Return self.u and self.t.
        """

    def advance(self):
        """Advance the solution one time step."""
```

UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Examination in:	IN1900 — Introduction to programming with scientific applications
Day of examination:	Monday, December 18, 2017
Examination hours:	9.00 – 13.00
This examination set consists of 6 pages.	
Appendices:	None
Permitted aids:	None

Make sure that your copy of the examination set is complete before you start solving the problems.

- This part contains suggested solutions for all questions except the multiple-choice questions.
- Note that for all the programming questions there are several possible solutions. The solutions presented here are usually among the simplest options, but not necessarily the simplest or the best. There are usually many alternative solutions that would also be given full score on the exam.

(Continued on page 2.)

Question 1.6

Answer:

```
[2, 3, 5, 7, 11]
[1, 2, 2, 4]
```

Question 1.12

Answer:

```
Hello, students
Hello, world!
```

Question 1.13

Answer: Nothing is printed, because the test passes.

Question 1.14

Answer:

```
{1: 2, 3: 1, 2: 2}
```

(Continued on page 3.)

Question 2.1

Solution:

```
def heaviside(x):
    if x < 0:
        return 0
    elif x>=0:
        return 1
```

Question 2.2

Solution:

```
def test_heaviside():
    tol = 1e-10
    x = (-1.0, 1.0)
    e = (0.0, 1.0)
    for x_,e_ in zip(x,e):
        assert abs(heaviside(x_)-e_)<tol, "Error for x = %g" %x_
```

Question 2.3

Solution:

```
from math import sin, pi

def diff(f,x,h):
    df = (f(x+h)-f(x))/h
    return f(x), df

#example use case
print(diff(sin,pi/2, 1e-6))
```

(Continued on page 4.)

Question 2.4

Solution:

```
from math import cos

for i in range(4):
    h = 0.5*1/(2**i)
    x = pi/4
    error = abs(diff(sin,x,h)[1]-cos(x))
    print('%6.3f %6.3f %6.3f' %(x, h, error))
```

Note that the text for Question 2.4 contained a typo. Students were asked to compute the error as the difference between the numerical derivative and the analytical result $-cos(x)$, but the derivative of $sin(x)$ is of course $cos(x)$. Naturally, answers were given the same score regardless of whether they used $cos(x)$ or $-cos(x)$.

Question 2.5

Solution:

```
from math import sqrt

def is_prime(k):
    if k == 1:
        return False
    else:
        for i in range(2,int(sqrt(k))+1):
            if k%i == 0:
                return False

    return True

#example use case:
for j in [1,2,7,9,13,27]:
```

(Continued on page 5.)

```
print(j,' ', is_prime(j))
```

Note that, for efficiency, the loop in this function only checks numbers up to the square root of k, since this is sufficient. The simplest solution is to check all numbers up to k, as described in the question text. This is less efficient, but was also given full score on the exam.

Question 3.2

Solution:

```
def solve(self):
    """Solve the ODE from time 0 to T.
    Store solution in self.u.
    Return self.u and self.t.
    """
    self.u[0] = self.U0
    for k in range(len(self.t)-1):
        self.k = k
        self.u[k+1] = self.advance()

    return self.u, self.t


def advance(self):
    """Advance the solution one time step."""
    f, t, u, k = self.f, self.t, self.u, self.k
    dt = t[k+1]-t[k]
    return u[k] + dt*f(u[k],t[k])
```

Question 3.3

Solution:

(Continued on page 6.)

```
class Heun(ForwardEuler):
    def advance(self):
        f, t, u, k = self.f, self.t, self.u, self.k
        dt = t[k+1]-t[k]
        u_star = u[k] + dt*f(u[k],t[k])
        return u[k] + 0.5*dt*(f(u[k],t[k])+f(u_star,t[k+1]))

#example use case
solver = Heun(f,1.0,10,100)
u,t = solver.solve()
print(t[0:-1:5],u[0:-1:5])
```

Question 3.3

Solution:

```
def predator_prey(x0,y0,r,a,m,b,T,n):
    def f(u,t):
        x,y = u
        return r*x-a*x*y,-m*y+b*x*y

    solver = Heun(f,[x0,y0],T,n)
    u,t = solver.solve()
    return t, u[:,0], u[:,1]

t,x,y = predator_prey(1,1,r=1,a=.3,m=1,b=.2,T=20,n=100)

import matplotlib.pyplot as plt
plt.plot(t,x,t,y)
plt.show()
```

END