tentamen

Tentamencode: BFVP15INF2

Fenna Feenstra en Jurre Hageman

30/01/2017

Inloggen/bestanden

Voor dit tentamen moet je ingelogd zijn als gast (password gast). Op de Desktop staan de volgende bestanden/links die je nodig hebt of kunt gebruiken om dit tentamen te maken:

1. een link naar HTML PyDoc
2. de Python cheat sheet
3. het script submit_your_work waarmee je de gemaakte opgaven inlevert.
4. de bestanden opgave1.py, opgave2.py en opgave3.py die je moet verbeteren. Maak een kopie van deze bestanden voordat je ze verandert (ze zijn read-only) en voeg je studentnummer toe aan de filenaam: opgave1_123456.py.
5. De file fasta.txt (nodig voor opgave 3).

Hulpbronnen

- HTML PyDoc opent de Python documentatie in een webbrowser
- Je mag gebruik maken van de Python cheat sheet
- In een terminal kun je pydoc aanroepen met bijvoorbeeld: pydoc print. Je krijgt dan een overzicht van de print functie
- In IDLE3 kun je de functie help() aanroepen. Voorbeeld: help(print) geeft je een overzicht van de print functie.

Opgaven

Het tentamen bevat een aantal opgaven. Iedere opdracht resulteert in een apart Python file. Begin elk script met de specificatie van de python3 interpreter (hashbang). Schrijf daarna je naam in het script, als metavariabele.

TIP: Programmeer per stap en kijk steeds wat er gebeurt. Gebruik IDLE3 of Python3 in de terminal om stukjes code uit te proberen.

Indienen

Het gemaakte werk kan ingeleverd worden door het script submit_your_work uit te voeren in de directory waar je het werk gemaakt hebt, als:

```
~/Desktop/submit_your_work STUDENTNR opg1.py opg2.py ...
```

Geef alle filenamen gescheiden met een spatie. Hierna kun je bij de surveillant controleren of het indienen gelukt is.

Beoordeling

Bij elke opgave staat vermeld hoeveel punten maximaal te krijgen zijn. De opgaven worden beoordeeld op de volgende punten:

1. Werkt de code?
2. Doet de code wat het moet doen?
3. Is de code goed (leesbaar, effectief)?

Als een stuk code niet werk, maak er commentaar van en schrijf er in commentaar bij wat je bedoelt.

Het cijfer wordt als volgt bepaald: behaald aantal punten/totaal aantal punten*9+1

LEES DE OPGAVEN GOED DOOR! Succes!

Opgave 1 [30 punten]

Opgave 1 is te vinden als opgave1.py. Pas deze file aan op basis van onderstaande opgave. Let op: de output van de code hieronder begint steeds met ##.

a) Print je voornaam (Engels: given name) en je achternaam (Engels: surname) met behulp van de print_name functie. Gebruik daarbij de functie print_name zoals gegeven in onderstaande code. Verander de volgorde van de parameters in de functie en de aanroep niet! Maak gebruik van keyword argumenten om je voornaam en achternaam in de juiste volgorde te printen. [5 punten]

```
first_name = "hier je voornaam"
last_name = "hier je achternaam"


def print_name(surname, given_name):
    print("Dit zijn de antwoorden van", given_name, surname)


print_name(first_name, last_name)

## ('Dit zijn de antwoorden van', 'hier je achternaam', 'hier je voornaam')
```

b) Herschrijf de functie print_all(a,b,c,d) naar een functie die oneindig veel argumenten kan printen. [5 punten]

```
def print_all(a,b,c,d):
    print(a, end=' ')
    print(b, end=' ')
    print(c, end=' ')
    print(d, end=' ')

print_all("1B:", 2,"primes", [251,257])
```

c) Herschrijf de onderstaand primer_list functie zodat het een generator function wordt die random DNA letters (A, T, C of G) genereert. De generator functie moet dit een aantal keer herhalen gelijk aan de primerlength. Dus als de primerlength 10 is moet er 10 * een DNA letter genereerd worden. [5 punten] Vervolgens moet de manier waarop de functie aangeroepen wordt gewijzigd worden om de generator functie aan te roepen. Gebruik hiervoor een for loop [5 punten]. Type de code niet over maar gebruik hiervoor de invulcode bij 1C uit opgave1.py

```
# originele primer_list functie die omgezet moet worden
def primer_list(primerlength):
    """
    The generate_primer function generates random primers
    """
    dna = "ATCG"
    primer = []
    for base in range(primerlength):
        random_base = random.choice(dna)
```

```
        primer.append(random_base)
    return "".join(primer)

print(primer_list(10))
```

Invulcode:

```
#vul de onderstaand code aan zodat generator_primer
#een generator function wordt [5p]
def generate_primer(primerlength):
    dna = "ATCG"
    #genereer hier een DNA letter (doe dat een primerlength aantal keer)

#programmeer op deze regel een for loop die de generate_primer functie
#aanroept met de primerlength
    print(base, end='')
```

d) Schrijf een list comprehension waarbij:

- RNA in DNA omgezet wordt (U naar een T) voor elke sequentie die begint met een A. Als je deze opdracht voor een deel werkend krijgt geef dan ook je antwoord. Je kunt hier punten voor krijgen. [10 punten totaal]

De volgende lijst:

```
rna = ["AUCGUC", "CGUCA", "GCAUC", "ACCUUGC", "CACACUU", "UUCAG", "AGGGG"]
```

Moet na uitvoer van je list comprehension als antwoord geven:

```
['ATCGTC', 'ACCTTGC', 'AGGGG']
```

Opgave 2 [30 punten]

Opgave 2 is te vinden als opgave2.py. Pas deze file aan op basis van onderstaande opgave.

In de file opgave2.py vind je een programma dat een commandline argument opvangt, controleert of het commandline argument wel DNA is en het GC gehalte berekent van de opgegeven DNA streng.

De Python file van opgave 2 werkt maar is niet netjes geschreven. Maak van dit script een net script. Werk de volgende stappen af.

1. Vul docstrings in waar nodig
2. Zet de auteur en het versienummer van je script in metavariabelen
3. Als de gebruiker geen argument op de commandline geeft dan moet het programma de documentatie weergeven
4. Maak vervolgens twee functies: check_if_dna en calculate_gc_percentage die de hieronder gegeven code gebruiken
5. De functie check_if_dna moet een boolean teruggeven
6. De functie calculate_gc_percentage moet een float (het gc percentage) teruggeven

7. Zet de functies check_if_dna en calculate_gc_percentage in een aparte module en noem deze: dna_tools.
8. Gebruik nu je dna_tools in je main module (opgave2.py).
9. Definieer in de main module een main functie en laat deze alleen uitvoeren als de module daadwerkelijk als main module gerund wordt.

De code van opgave 2 vind je dus in opgave2.py maar is hier ook gegeven. Dit hoef je dus niet over te typen.

```
#!/usr/bin/env python3

#imports
import sys

#get arguments
args = sys.argv[1:]
#get sequence and make upper case
seq = args[0].upper()

#check if string is dna:
dna = "ATCG"
for base in seq:
    if not base in dna:
        print("Found invalid character in DNA sequence {}".format(seq))
        print("Program will exit...")
        sys.exit()

#calculate the GC percentage
base_G = seq.count("G")
base_C = seq.count("C")
len_seq = len(seq)
perc_GC = ((base_G + base_C)/len_seq) *100

#print the output
print("The GC percentage of sequence {} is: {}".format(seq, perc_GC))
```

Opgave 3 [30 punten]

Schrijf een programma dat in de main functie twee argumenten opvangt van de commandline: de fasta.txt en de naam van een outputfile. Print de documentatie van de module als niet het juiste aantal argumenten worden meegegeven op de command line. Gebruik hiervoor exception handling. De inhoud van fasta.txt moet regel voor regel worden weggeschreven met een index voor elke regel. Programmeer deze filebewerking in een aparte add_index functie. Gebruik de ...with open... constructie en gebruik exception handling in de add_index functie.

```
#!/usr/bin/env python3
"""
In the main() function a name for the inputfile(fasta.txt) and the outputfile
(out.txt) are fetched from the commandline [5 points]

the main function checks whether the correct number of arguments are parsed
on the command line
if not, the function calls the documentation of this module [5 points]

In the add_index(..,..) function the inputfile and the outputfile are opened
with automatic closure construction [5 points]

each line from the inputfile is read and written to the outputfile with an
index prefix (starting with 1) using the enumerate construct [5 points]

example input line from inputfile:     "CAAGAGAGCTGAGCTTGGA"
example output line outputfile:      "1 CAAGAGAGCTGAGCTTGGA"

the add_index function handles the IOError in case the inputfile does not
exist or the outputfile could not be created [5 points]

"""



#imports
import sys
import pydoc


#defs
def add_index(): #change this
    """
    read a file and write the content to a new file with linenumber prefix
    :param input: file that needs to be read (for example fasta.txt)
    :param output: file that needs to be written to (for example out.txt)
    """
    #open the files and close automatically
    #each line of input needs to be write to output
```

```
    #with linenumber prefix using enumerate
    #in case of an IOError, catch the error and print a message


def main():
    """
    fetch from commandline the name of the inputfile and outputfile and parse
to add_index
    use exception handling when no input file and output file are given
    """
    #fetch from commandline input and output and parse to add_index
    #in case of an IndexError call pydoc.help and exit


#call main

                        --- einde tentamen ---
```