

ФИО и почта препода: Александр Бобровских, avb@bionet.nsc.ru

Вводная часть

Ключевые слова начала занятия: Большие массивы данных для предобработки транскриптомных данных, Уровень экспрессии генов, Выявить дифференциально экспрессирующиеся гены.

Предыстория: Выделили РНК, отдали на секвенирование в Китай в геномный центр, вам в fastqс пришли данные, вы хотите дальше с ними работать. Вам поможет знание программирования, командной строки, настойчивость разобраться в том, почему сначала не поставилась определенная библиотека или модуль. **Знания Питона или R облегчает рутинную процедуру.**

Программы: SciPi, Pandas – для работы с большими массивами данных; **Профессии:** аналитики данных

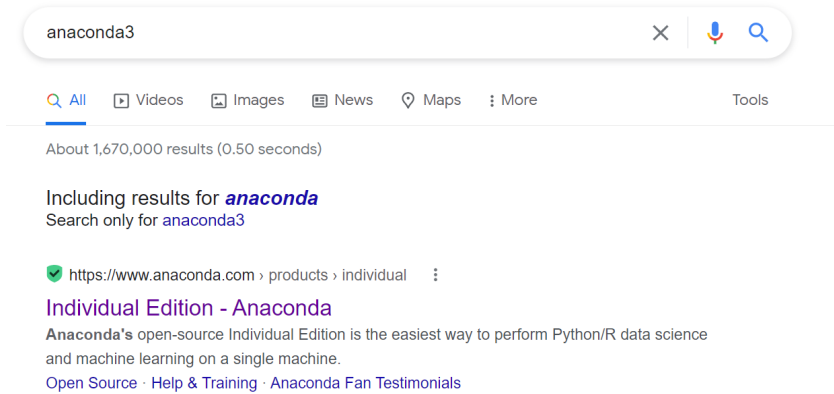
Основное: Питон: питоне есть модуль отдельный: один из них предназначен для того, чтобы сделать жизнь биоинформатиков проще: BioPython - в нем же реализованы методы для удобной работы с последовательностями, с доступом к psbi – полезная библиотека. Когда с типичными форматами био -данных работаете. Питон не требует больших временных затрат, объемов знаний, подходит новичку; и есть продвинутые библиотеки и т.п.

Питоны между собой: между собой по синтаксису базовому слегка отличаются (например, нужны скобки или нет)

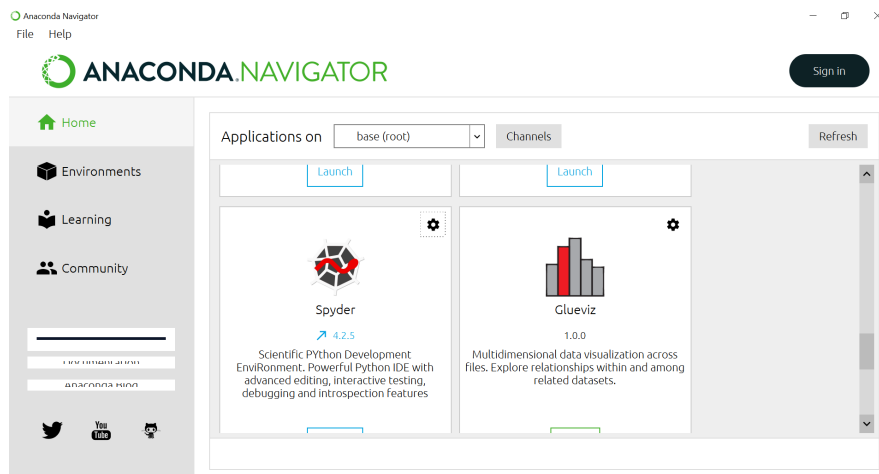
Алгоритм этапов разработки (и на питоне, и не только): анализ, проектирование, реализация (решили на GitHub добавить и обновляете иногда), тестирование, внедрение, сопровождение

Практикум: как сгенерировать несколько последовательностей ДНК или РНК и посчитать их GC состав? (в spyder)

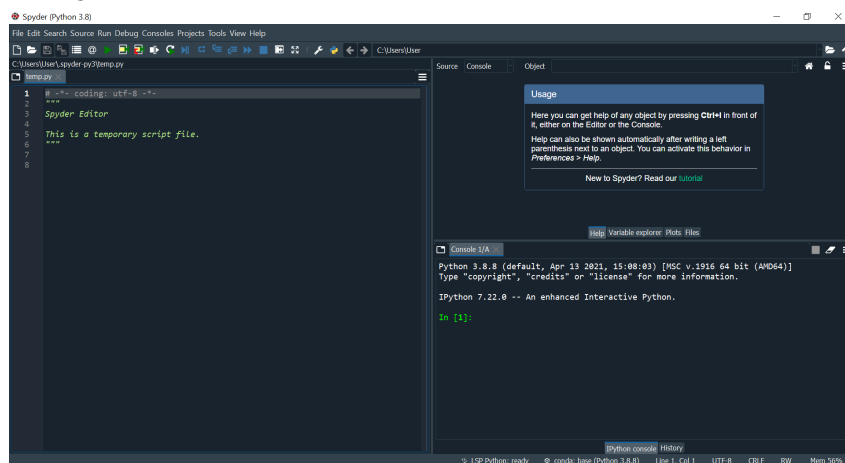
1) Поставили анаконду:



2) В anaconda.Navigators находим spyder:



3) Spyder:



4) Освоим базовые вещи для работы:

- **Поговорим про имена переменных:**

Например: При названии переменных не надо начинать с цифр.

```
1 s23 = 1
```

Вот так можете называть переменные

```
1 23s = 1  
2
```

А вот так нет. С цифры название не начинается в питоне

- **Хочу узнать тип переменной какой-то:**

value2 - это моя переменная. И вот так можно узнать ее тип:

```
value1 = 1  
value2 = 2  
value3 = value1/value2  
print(type(value3))
```

Ответ:

```
<class 'float'>
```

- **Хочу вывести значение:**

```
value1 = int(1)  
value1 = float(1)  
print(value1)
```

ответ:

```
1.0
```

- Есть классическое деление слэшом: /
- Есть функция узнать остаток от деления: это %

```
del_ost = 2%value2  
print(del_ost)
```

и ответ:

```
0
```

- **Хочу узнать тип переменной:**

```
seq = "AATATGCGCGTGT"  
print(type(seq))
```

```
<class 'str'>
```

- Допустим, есть уровень экспрессии каких-то генов. Создаем массив для хранения:

```
seq2 = "ATN, 20, 50, 60, 70, 100"  
list_new = seq2.split(", ")  
print(list_new)
```

и он может сделать так:

```
['ATN', '20', '50', '60', '70', '100']
```

(Split позволяет более

строго работать с элементами, так удобнее)

- а вот я хочу длину посмотреть: сколько у меня элементов?

```
print(len(list_new))
```

ответ: 6

- теперь я хочу первый элемент достать из массива:

первый элемент в листе - нулевой = индексация начинается с нуля (это иногда может запутать)

```
print(list_new[0])
```

ответ: ATN

- достаю первые два элемента:

```
print(list_new[0:2])
```

ответ:

```
['ATN', '20']
```

- Можно сравнивать срезы листов, например для сравнения групп генов по уровню экспрессии:

```
l1 = list_new[1:3]  
print(l1)  
l2 = list_new[3:]  
print(l2)
```

ответ:

```
['20', '50']  
['60', '70', '100']
```

- Проверим снова тип переменной:

```
print(type(l1[0]))
```

Ответ:

```
<class 'str'>
```

- это строка, с ней у вас не выйдет делать арифметические операции.

- Я могу посчитать сумму элементов массива:

```
import numpy as np  
  
arr1 = [1, 2, 3]  
print(sum(arr1))
```

ответ:

```
6
```

- А вот я хочу все элементы из какого-то множества вывести в случайном порядке:

```
for element in range (0, 10):
    print(element)
```

ответ: 6
0
1
2
3
4
5
6
7
8
9

- Хочу увидеть только последний элемент при просмотривании всех элементов множества:

Отступ имеет большое значение: цикл пробежал все элементы и запомнил последнее значение - его и выводит:

```
for element in range (0, 10):
    value = element
print(value)
```

ответ: 9

- А теперь создам словарь:

```
slovar = {}
```

- Проверим его тип:

```
print(type(slovar))
```

ответ: <class 'dict'>

- Удобно, если например вам нужно посчитать, сколько вхождений элементов в ваш файл входит: например, сколько раз в текст входит слово "последовательность"

```
slovar["vals1"] = 3
print(slovar)
```

ответ: {'vals1': 3}

- то же самое можно сделать по-другому:

```
print(slovar["vals1"])
```

ответ: 3

- А теперь я хочу арифметически поработать со значениями внутри “словаря”:

```
slovar["vals1"] = slovar["vals1"] + 3
print(slovar)
```

ответ:

```
{'vals1': 6}
```

- Аналогично можно складывать, но просто записывать это по-другому:

```
slovar["vals1"] += 3
print(slovar)
```

в ответе соответственно получили значение, увеличенное

```
{'vals1': 9}
```

на 3:

Вопрос. Как я могу применить это в биоинформатике? А вот я хочу посчитать, сколько у меня аденинов в ДНК:

```
seq3 = "ATTTTGACTAC"
a_numb = seq3.count("A")
print(a_numb)
```

ответ:

```
3
```

5) Биоинформатика. Задачи

Сгенерировать 100 последовательностей ДНК (случайных) длиной 60 п.н.

```
import random as rd
alphabet = ["A", "T", "G", "C"]
symb = rd.choice(alphabet)
i = 0
list_seq = []
while i < 100:
    currseq = ''
    while len(currseq) < 60:
        currseq = currseq + rd.choice(alphabet)
    i += 1
    list_seq.append(currseq)
print(list_seq)
print(i)
```

Решение:

Суть решения: для генерирования строки длиной 60 создаем пустую строку currseq. С ней происходит следующее: увеличиваем текстовую последовательность до тех пор, пока ее длина не станет равна 60. Ставим счетчик на количество созданных последовательностей длины 60. Когда он станет равным 100 - нужно перестать генерировать последовательности.

```
[ 'CCCTGAACTAGAAAATTGTCGGAAGTACGACGCGTATACGCATCGGACTGTGACGCAAGAA',
'GAAGAGCTCGACGGAAGAGCTTGCAGCTCGTGCTAGTACTCTCGTTAGGCACCGTGTAGC',
'TCTAAACCTGTTTTAATTTTACATCGGGCGCAGCATGCTCGTTATCCACCCCTCACTGGT',
'CTAGCACTGAACGCGCGCGTGAAAAATGTAGCTGTATCAACTTGAACCCCGTACCGCCAAC',
'GATTATAGTGCGAAGGCTGTAGATTGTACTATACACTCGCACGGCAGCGCCTTGGATCCG',
'TAGTAGCCCAAGCCAGTAGAATTGTTTAGAGGCCGTATCGGAAAAGGCTAGAGGACGCAA',
'GGGTGAACATCTCGAATAGGGTTTGCCCGGTCTGAGTCCTAGTGAGTGGTTGATTAAG',
'GGGGTAAAGCAAAGGAAGAGTCCCTGAACGGCCAAGGACCCCATATACCTCAAGGGAGA',
'ATGAAGCATGTGGAGTGTGAAGGCCGTCCACCGTAAGAAAACGGCCTCCACGCATGAT',
'ACGACTTCTGGATAAGTCACGACATTAGTCCTGATGATGCGGCTTGTGAAAGTAGCTGA',
'TGCAAACTTAGATTGCTTTCGGTTTTTTTATTCCCTGCCGTACCTAGCAAGTTCAGACG',
'AGACCGCGTCAATTACAGGCCGGGTCAAGCTAACGCCCTCACCAGAGGCACTTTCATT',
'TCAGGAACCACTCGTCCCTTCAACGGCCGGTCCGGGGCCGATGTGCTGGGTGTCTCATG',
'ATATTAGATTCAATTGCAACATCCTCCAGCCTGACTCCCTTCTACAAGCCGAAGGAGT',
'AGAGGCTGGCTCATTTGCGAACCCCTATTTCGGTGCCTCGACGATGGTATAAGTCTTAC',
'CGCATTCGCCCGCAGCTAAACGGGAACATATCCCATCCATTCAATACGAGTCGTTTTA',
'CAGGGCGCATACTCGAATTGCGACTTTAAATGGTTAGGATTCACTCACGTCACCGCTGTA',
'GCCCTGTGCAAGTCGGTAGCAACGGTAACATTGCTTTGTAGAAGAAGCAGGCTTTACTA',
'AATTAGGCCAGGTGCAACTTATAGGCTGCGTATACTTTACGCCGATATCGGTAGCATAAT',
'CATTGAAGGTGTCAAGTAATGTAGACGGTGCGGTAAGTTTTAATCGCGTTGTAGAGATT',
'CTCACTCTTGTAAGGAATTGCTGTGAAGAGCTGGGTGGGCATTCTACCGGAATGAACCAC',
'ACCAATAAAATTTAACATCGGGTTTCGACAGGAGGCGTCTCTAGATCCAGACTGACTGGA',
'TATCATTAGAGCTGACGGCAATCGGGGCTGCAAGATGTTGCTGTCGTTCTCATCTCGCT',
'ATCTTCGTATGGCAGGATCTAGTATCATTCCCATTTGATTGAGTTCCAAGGACAGGGTG',
'CTGTGATCTGACGTGGGTAGGGCTAATTTTGCAAACCTCGCCTACTGTGCCTAAAAATAG',
'GTTCCATCAGCGCACGACTCATATAACTTTTGAGACAAGCCATGGATCATGTCAAGTCA',
'CGCTAACTCCCTTGGGTGTTTCAGGCAAAAGTTCTCCTGATCCGGTGACCCGTATATGC',
'CACTAAGTCTAGTAAATATTACGTCTCACGGCGACTAAATCCGACGGCGATGCTGACTTAG',
'ATTTGGCCAACATTTGATTTTCGTTGCGAATTTCCAACAGGCCCGCAACATAATTATG',
'CACCAAACAGCTTGCGCAGACCGGGCTTCTGATGAGTACCGACCTTAGATCACCTTACAT',
'CTATCTCATTTGTACGATTGCAAGCCAGTTGATTTAAGAGTCCGGATCCGCCTCTCTCGAT',
```

- здесь 100

последовательностей длины 60.

Проверим, что длина каждой последовательности 60. Так как они равноправны, выведем размер любого элемента, например самого первого:

```
print(len(list_seq[0]))
```

ответ:

60

Если я хочу сделать последовательность увеличивающейся длины:

```
import random as rd
alphabet = ["A", "T", "G", "C"]
symb = rd.choice(alphabet)
length = 1
i = 0
while i < 100:
    while length < 60:
        symb = symb + rd.choice(alphabet)
        length += 1
        print(symb)
        print(len(symb))
    i += 1
```

Определить GC состав каждой последовательности из массива

Чем больше GC пар, тем более тугоплавка последовательность.

Внесу изменения в написанный код:

```

import random as rd
alphabet = ["A", "T", "G", "C"]
symb = rd.choice(alphabet)
i = 0
list_seq = []
GC_list = []
while i < 100:
    currseq = ''
    while len(currseq) < 60:
        currseq = currseq + rd.choice(alphabet)
    GC_curr = (currseq.count("G") + currseq.count("C")) / 60
    i += 1
    list_seq.append(currseq)
    GC_list.append(GC_curr)
print(list_seq)
print(i)
print("\n")
print(len(list_seq[0]))
print("\n")
print(GC_list)

```

Я создаю еще один массив, в который будут записываться элементы, представляющие собой GC состав каждой созданной последовательности. Для этого каждый раз после создания последовательности длиной 60п.н. я считаю ее GC состав. И добавляю это значение в массив, который будет хранить GC состав каждой последовательности. Получился массив, хранящий GC состав каждой из 100 последовательностей длиной 60:

```

[0.55, 0.48333333333333334, 0.5666666666666667, 0.65, 0.5833333333333334,
0.48333333333333334, 0.4, 0.5166666666666667, 0.5666666666666667, 0.43333333333333335,
0.5166666666666667, 0.4666666666666667, 0.48333333333333334, 0.5, 0.5333333333333333,
0.45, 0.45, 0.5166666666666667, 0.5333333333333333, 0.5333333333333333,
0.5833333333333334, 0.4666666666666667, 0.35, 0.5833333333333334, 0.4666666666666667,
0.4833333333333334, 0.4833333333333334, 0.4833333333333334, 0.55, 0.43333333333333335,
0.5666666666666667, 0.5, 0.5, 0.5833333333333334, 0.45, 0.55, 0.5166666666666667, 0.5,
0.5666666666666667, 0.6, 0.6, 0.5333333333333333, 0.4166666666666667, 0.43333333333333335,
0.4666666666666667, 0.4833333333333334, 0.4833333333333334, 0.38333333333333336,
0.6166666666666667, 0.45, 0.6166666666666667, 0.5333333333333333, 0.55,
0.5666666666666667, 0.55, 0.4833333333333334, 0.5, 0.5333333333333333,
0.4666666666666667, 0.5166666666666667, 0.45, 0.5, 0.5, 0.6166666666666667, 0.45,
0.5833333333333334, 0.5166666666666667, 0.5, 0.5, 0.6333333333333333, 0.5,
0.5166666666666667, 0.5833333333333334, 0.5333333333333333, 0.5333333333333333, 0.5,
0.5166666666666667, 0.38333333333333336, 0.6333333333333333, 0.6166666666666667, 0.55,
0.5833333333333334, 0.6166666666666667, 0.5, 0.5, 0.5333333333333333, 0.4666666666666667,
0.55, 0.45, 0.4666666666666667, 0.5666666666666667, 0.55, 0.4833333333333334, 0.5,
0.6333333333333333, 0.4, 0.5333333333333333, 0.5166666666666667, 0.5666666666666667, 0.45]

```

Теперь я хочу округлить все числа до 3 знаков после запятой:
Вношу изменения ТОЛЬКО в эту строчку:

```

GC_curr = round((currseq.count("G") + currseq.count("C")) / 60, 3)

```

И получаю:

```

[0.517, 0.517, 0.533, 0.517, 0.517, 0.5, 0.483, 0.6, 0.417, 0.6, 0.517, 0.467, 0.467,
0.383, 0.433, 0.483, 0.517, 0.467, 0.45, 0.517, 0.4, 0.467, 0.617, 0.533, 0.567, 0.583,
0.417, 0.533, 0.517, 0.6, 0.55, 0.45, 0.467, 0.433, 0.6, 0.467, 0.45, 0.55, 0.417, 0.567,
0.55, 0.583, 0.467, 0.433, 0.517, 0.5, 0.483, 0.4, 0.4, 0.6, 0.5, 0.433, 0.65, 0.45,
0.517, 0.517, 0.5, 0.5, 0.383, 0.567, 0.633, 0.517, 0.367, 0.467, 0.467, 0.55, 0.6, 0.533,
0.417, 0.517, 0.467, 0.533, 0.4, 0.383, 0.367, 0.583, 0.483, 0.383, 0.433, 0.55, 0.617,
0.5, 0.483, 0.5, 0.567, 0.467, 0.583, 0.5, 0.483, 0.533, 0.55, 0.483, 0.567, 0.617, 0.45,
0.417, 0.5, 0.5, 0.483, 0.55]

```


Снова теория

- Создаем файл:

```
for element in list_seq:
    file.write(">seq_ " + str(i) + "\n")
    file.write(element+"\n")
    i += 1

file.close()
```

Теперь в spyder где-то создан файл.

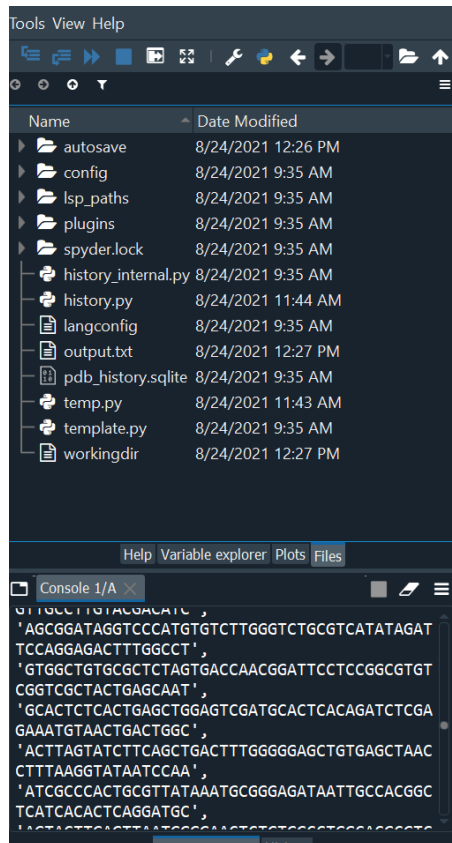
- Открываем файл:

The screenshot shows the Spyder IDE interface. The top panel displays the Variable explorer with the following variables:

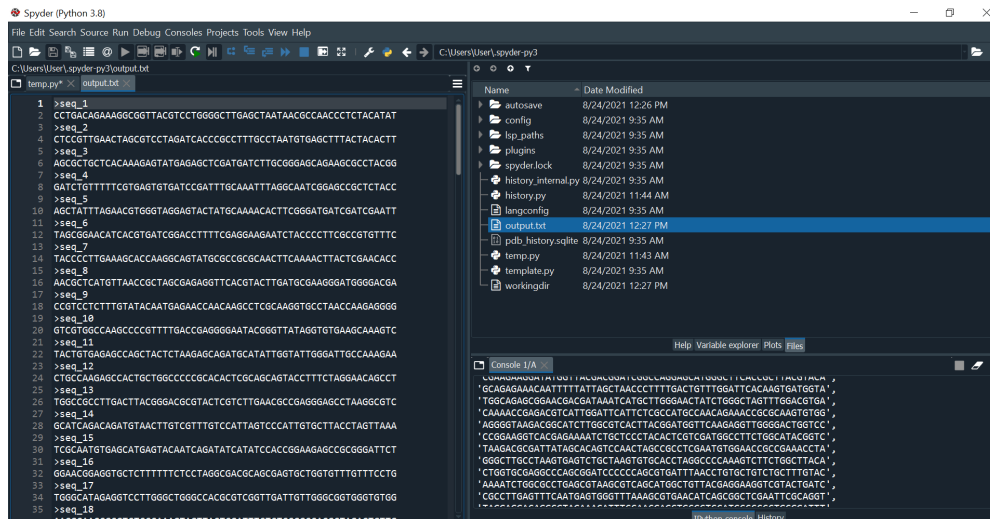
Name	Type	Size	Value
a_numb	int	1	3
alphabet	list	4	['A', 'T', 'G', 'C']
arr1	list	3	[1, 2, 3]
currseq	str	60	CAGCCCTGGCGGCTACTCCGACGACAGATGATCTCAATGATGCCCCGGTTCGTCT
del_ost	int	1	0
element	str	60	CAGCCCTGGCGGCTACTCCGACGACAGATGATCTCAATGATGCCCCGGTTCGTCT
file	TextIOWrapper	1	TextIOWrapper object of _io module
GC_curr	float	1	0.583
GC_list	list	100	[0.517, 0.483, 0.567, 0.467, 0.417, 0.517, 0.517, 0.533, 0.517, 0.55, ...]
i	int	1	101

The bottom panel shows the IPython console with a large block of DNA sequence data. A yellow circle highlights the 'Files' tab in the bottom right corner of the interface.

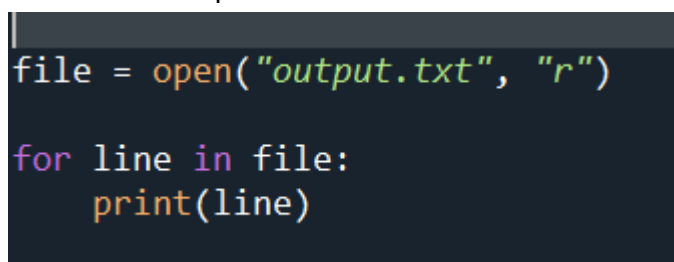
- Заходим в “files”:



Наш файл называется “output.txt”:



- Чтение файла:



- что я вижу в окне console:

```

>seq_10
GTTACCAACCGTTGCAGCCCCCCCAGCACGTGCAGATCACACTTATAGGGTTCCAT

>seq_11
CAGCGGCGCTTCAACGGTCATGGCTCTGACGCGGTATGCGTCGGTATCACGGCATGCC

>seq_12
ATTAGAGAGATGTCTCGTAGATGCATCATTGGAGACCCGTCTGATGTAATAATGAATGGT

>seq_13
AACACAGTCCTGGGTATGCTTCAATTCGGCCGTCGGCTCGGTCGCTCAAACAGCGGAGG

>seq_14
GCCGGGGGTTTCAAGTAGTAGGTCGTAACCAATAAAATCACTGTTGGTGCCATAAAGTGA

>seq_15
GGTCAGTCGCGCTCGTTTTGACCTAAATGTGCGGACGGTCCGGCCAGCCACAGAAAGTG

>seq_16
TTATGATCACAATGCGAGCGGGTCAAGAATCACGCCTATATTAACCAATGATGGGTGCC

>seq_17
CGTCTCATCTGATATAATGTAACATTAGAACTCCGTCAAGATGCGATGTCATTGTAGCT
  
```

- Функция, которая умеет считать GC состав:

```

def GC_content(dna):
    g_comp = dna.count("G")
    c_comp = dna.count("C")
    gc_content = (g_comp + c_comp)/len(dna)
    return(gc_content)
  
```

Например:

```

dna = "TAACTCATGGGCGTGAT"
print(GC_content(dna))
  
```

ответ: 0.47058823529411764

А теперь я хочу получить ответ с точностью до трех знаков:

```

dna = "TAACTCATGGGCGTGAT"
print(round(GC_content(dna), 3))
  
```

ответ: 0.471

- А если я скачала fasta файл с последовательностями и хочу в них GC состав посчитать? То то же самое пишете:

```
file = open("output.txt", "r")
```

той же папке, что и ваш файл.

- но это только если скрипт в

- А если скрипт в одном месте, а папка в другом? Можете прописать путь и название файла: в файле open прописать путь до файла через двойные слэши - если windows (это где file = open("путь", "r"))

Попытки собрать код для вычисления GC состава в файле, который мы сами выбрали:

```
file = open("output.txt", "r")

for line in file:
    line1 = line.replace("/n", "")
    check_seq = line.find(">")
    if check_seq == -1:
        GC_content = GC_content(str(line1))
        print(GC_content)
    else:
        print(line)
```

проблема:

```
TypeError: replace expected at least 2 arguments, got 1
```

логика: в файле строка сиквенсы и название. Пропускаем строки, которые начинаются со знака ">", а остальные считываем и считаем GC состав. Метод find является проверкой. Если в строке нет нужного символа - он возвращает "-1". Если находит - возвращает координату, начиная с нулевой. Строка содержит название сиквенса - она нам не нужна. Так что условие выбираем if else. Выводим имя последовательности в одном случае, а во втором считаем GC состав.

А зачем герласе? Такая структура данных, что к любой строке добавлен /n в конце. И чтобы его не считать в длине, то стоит убирать.

Рабочий код для выполнения задачи вычисления GC состава в нашем файле:

```
file = open("output.txt", "r")

for line in file:
    line1 = line.replace("/n", "")
    check_seq = line1.find(">")
    if check_seq == -1:
        g_comp = line1.count("G")
        c_comp = line1.count("C")
        gc_content = (g_comp + c_comp)/len(line1)
        print(gc_content)
    else:
        print(line1)
```

что я вижу в окне console:

```
<function GC_content at 0x00000196564645E0>
>seq_92

<function GC_content at 0x00000196564645E0>
>seq_93

<function GC_content at 0x00000196564645E0>
>seq_94

<function GC_content at 0x00000196564645E0>
>seq_95

<function GC_content at 0x00000196564645E0>
>seq_96

<function GC_content at 0x00000196564645E0>
>seq_97

<function GC_content at 0x00000196564645E0>
>seq_98

<function GC_content at 0x00000196564645E0>
>seq_99

<function GC_content at 0x00000196564645E0>
>seq_100

<function GC_content at 0x00000196564645E0>
```

Дополнительно

Объектно-ориентированное программирование:
Создавать объекты определенного класса можно

Сами классы создаются - вы в явном виде объявляете класс
pass - пропустить, будет пустой класс

С помощью класса мы создаем свои новые типы объектов. Например, вес, рост, язык человека. Как в тех же РПГ есть класс: лучник, воин. В классе можно создавать соответственно определенные объекты, обладающие определенным набором свойств.