Київський національний університет імені Тараса Шевченка
ННЦ "Інститут біології та медицини"

Ярослав Ясінський, Андрій Сиволоб

Методичні рекомендації до лабораторного практикуму з генетики "Обробка генетичних даних за допомогою сучасних методів програмування"

для студентів освітньої програми "Біологія"
освітнього рівня "Бакалавр"

Київ – 2023

УДК 519.685:575.113.1:577.2

Рецензенти:

професор кафедри біофізики та медичної інформатики, д-р фіз.-мат. наук, професор Юрій Прилуцький

доцент кафедри вірусології, канд. біол. наук, с.н.с. Олексій Шевченко

Рекомендовано до друку вченою радою ННЦ "Інститут біології та медицини"
(протокол № 15 від 30 червня 2023 року)

Ясінський Я., Сиволоб А. Методичні рекомендації до лабораторного практикуму з генетики "Обробка генетичних даних за допомогою сучасних методів програмування" для студентів освітньої програми "Біологія" освітнього рівня "Бакалавр" ННЦ "Інститут біології та медицини", 2023. – 73 с.

Викладено описи практичних робіт, спрямованих на опанування основами програмування на мовах Python і R та їхнього застосування для аналізу біологічних даних. Представлено приклади такого застосування та практичні задачі, розв'язок яких має дати студентам уявлення про можливості вказаних програмних засобів. Описано також основні бази даних біологічної інформації та практичну роботу, що має не меті ознайомити студентів з такими базами.

Вступ

У сучасній генетиці безпосереднім результатом роботи дослідників часто є великі об’єми даних. У більшості випадків з напрацьованих даних дослідницька група обробляє невеликий відсоток і використовує та представляє їх частково. Використання новітніх методик лабораторного аналізу дозволяють пришвидшити отримання результатів, тому невикористаних і ще не проаналізованих даних генерується все більше.

Для раціонального використання результатів праці дослідників були створені сучасні підходи роботи з такою “сирою” інформацією – розроблені інструменти з її публікації, зберігання, забезпечення широкого доступу іншим дослідницьким групам.

Практикум має на меті на конкретних практичних прикладах ознайомити з деякими сучасними інструментами доступу до інформації та її обробки, які широко використовуються в генетичних дослідженнях.

Під час курсу ви поглибите свої знання в підходах до автоматизації повторюваних процесів, ознайомитесь з синтаксисом і створюватимете власні скрипти на мові програмування Python, ознайомитесь з можливостями і окремими інструментами в оточенні RStudio для мови R, дізнаєтесь про бази даних біологічної, зокрема генетичної, інформації, познайомитесь з типами і архітектурою серверних баз даних.

В роботі використані програмні продукти, що розповсюджуються за відкритою або вільною ліцензією. Виконання практики доступно на більшості популярних операційних систем - Windows, MacOS, дистрибутивах Unix-систем. Програма курсу адаптована до дистанційного та асинхронного навчання.

Завдання практикуму мають різні рівні складності, щоб зробити курс цікавим для студентів з різним базовим рівнем ознайомлення з мовами програмування.

Складність позначається зірочками «*» Відсутністю зірочок позначаються завдання, виконання яких передбачає ознайомлення лише з даним курсом. Наявність одніє чи двох зірочок передбачає готовність студента шукати інформацію самостійно.

Практична робота №1

Тема: Методи створення алгоритмів. Встановлення програмного середовища для використання мови Python

Основним інструментом оптимізації обчислювальних процесів є створення алгоритмів. Основним підходом до створення алгоритмів є використання мов програмування.

Python є представником інтерпретованих мов програмування. Цей тип мов передбачає інтерпретацію коду по рядках - кожний рядок є окремою інструкцією, які виконуються почергово.

Для взаємодії з програмним кодом створені середовища розробки - програмне оточення, інструментами якого здійснюється виконання скриптів користувача. Воно є проміжною ланкою між графічним інтерфейсом і машинним кодом.

Conda є середовищем розробки для продуктів на мовах Python та інших. **Miniconda** - це полегшена версія середовища для використання з інтерпретатором Python.

Для виконання, зручного перегляду результатів та аналізу помилок існують інтерфейси взаємодії.

Для проектів цієї лабораторної роботи середовищем слугуватиме програмне оточення Miniconda3 і веб-інтерфейс взаємодії з ним **Jupyter**.

Мета роботи: Ознайомитись з основами мови Python. Підготувати оточення для виконання скриптів на цій мові.

Матеріали та обладнання:

- Компьютер з операційною системою Windows, MacOS або дистрибутивом Unix-системи.
- Доступ до мережі internet

Хід роботи:

1. Ознайомитись з загальною інформацією щодо мови програмування Python
2. Встановити Miniconda3 відповідно до вашої операційної системи
3. Запустити мініконду:
    a. Unix/MacOS - командою conda в терміналі

    b. Windows - Пошук "miniconda", оберіть Anaconda Prompt (Miniconda 3)
4. Почергово ввести наступні 7 команд. (Після кожної команди в терміналі буде відбуватись встановлення пакетів, що займає до хвилини часу)

```
conda update conda --yes
conda install ipython --yes
conda install pyzmq --yes
conda install jinja2 --yes
conda install tornado --yes
pip install notebook
jupyter notebook
```

5. На сторінці Jupyter, що відкриється після останньої команди з попереднього пункту, створити новий проект на мові Python 3
6. Перевірити працездатність розгорнутого проекту. Для цього внести арифметичну дію, наприклад “2+2” і натиснути Ctrl+Enter / Cmd+Enter
7. Зробити висновки щодо мети встановлення кожного програмного продукту (conda, jupiter), який був вами використаний.

Практична робота №2

Тема: Синтаксис Python. Написання і виконання скриптів. Змінні, математичні дії.

Python - це високорівнева, інтерпретована мова програмування, що відома своєю простотою в використанні та ефективністю для вирішення різних задач програмування, таких як веб-розробка, наукові обчислення, штучний інтелект, машинне навчання та багато іншого.

Інтерпретатор можна використовувати як калькулятор за допомогою простих команд Python. Доступні стандартні математичні оператори **+, -, *** та ***/***. Дужки можуть бути використані для групування і встановлення порядку дій.

Використовується декілька типів чисел, серед них найпоширеніші - цілі (integer, **int**) і з плаваючою комою (floating-point, **float**)

Якщо в розрахунках на якомусь етапі використовується число типу float, результат також буде float. Результат поділу - це завжди float.

Оператор ***//*** означає поділ з результатом в типі **int** – система поверне цілу частину від поділу. Наприклад

```
15 / 5 => 3.0 (float)
17 / 3 => 5.666666666666667 (float)
17 // 5 => 3 (int)
```

Оператор **%** означає залишок від поділу

```
17 % 5 => 2 (int)
```

Оператор ** означає зведення в ступінь

```
2 ** 10 => 1024 (int)
16 ** 0.5 => 4.0 (float)
```

Змінні - це імена, які використовуються для збереження значень. Присвоїти значення змінній можна оператором “**=**”

```
number = 10 # змінна number зберігає значення 10
result = number * 3 # змінна result зберігає значення 30
```

Рядки. Крім чисельних значень в Python можуть використовуватись рядкові (стрічкові). Рядкові значення вносяться в одинарних ('...') або подвійних лапках ("...")
Знак \ може бути використаний для екранування лапок (апострофів) - щоб лапка розумілась як символ, а не кінець стрічки

```
name = ‘Іван’ # змінна word зберігає значення Іван
quotes = ‘“Veni, vidi, vici” Юлій Цезар’ # змінна quotes зберігає значення “Veni, vidi, vici” Юлій цезар
word = ‘Кип\’яток’ # змінна word зберігає значення Кип’яток
```

З рядковими даними можна проводити математичні операції

```
name = ‘Іван’
greetings = ‘Вітаю, ‘
print(greetings * 3) # Вітаю, Вітаю, Вітаю,
print(greetings + name + ‘!’ * 3) # Вітаю, Іван!!!
```

При роботі з рядковими даними звернутись можна не до всього рядка, а до номерів символів у рядку, вказавши їх у квадратних дужках.

(!) Зверніть увагу, що в Python, як в більшості мов програмування, **нумерація елементів починається з 0**. Тобто перший символ буде мати індекс “0”, третій - “2”, тощо.

```
word = “Генетика”
word[1] # “е” (другий символ в рядку)
word[-1] # “а” (останній символ, перший з кінця)
word[1:3] # “ен” (символи з індексами, що більше або дорівнюють 1, але менше 3. Індекси “1”, “2” - другий та третій символи рядка)
word[2:] # “нетика” (від символа з індексом “2” до кінця рядка)
word[:6] # “Генети” (від початку рядку до символа з індексом “5”, тобто шостого)
```

Вхідні і вихідні дані. Робота програми передбачає отримання інформації від користувача, обробки алгоритмом і повертання відповіді

користувачу.

Команда `print` використовується, щоб вивести значення в інтерфейс користувача. За замовченням, в кінець додається символ зносу строки, тобто декілька команд `print` будуть виводити кожне значення з нової строчки. Щоб змінити поведінку, можна використати набір аргументів `print` наступним чином:

```
print('Привіт ', end='')
print('Іван!')
```

В цьому випадку після першого виведення значення на екран, символ зносу рядка не буде додано, і рядок буде «Привіт Іван!»

Для запиту інформації від користувача використовується команда `input()`. Коли інтерпретатор доходить до неї, він зупиняє обробку даних і чекає внесення даних від користувача.

```
word = input()
print(word*2) # інтерпретатор дочекається внесення рядка від користувача, потім виведе цей рядок двічі
```

Пояснення для користувача можна давати аргументом в команду `input()`

```
a = input('Введіть перше число: ')
```

Вхідні дані від користувача **завжди є рядковим значенням**.

(!) Зверніть увагу, що `'2' * 2 = '22'`, а `'2' + 2` – помилка типів даних `(TypeError)`.

Щоб отримати дані у вигляді числа, їх можна конвертувати командою `int()` або `float()`. Конвертація в рядок виконується командою `str()`

```
number = input() # число буде збережено рядком
number = int(number) # число буде збережено типом int
number = str(number) # число буде збережено типом str
number = int(input()) # число буде збережено типом int
```

Мета роботи: Ознайомитись з синтаксисом Python. Опанувати інструменти вводу та виводу інформації, використання математичних дій на мові Python.

Матеріали та обладнання: Оточення Miniconda3, інтерпретатор Jupyter

Хід роботи:

1. Ознайомитись з документацію щодо використання математичних операторів в Python
2. В оточенні Jupiter написати скрипти, що виконують інструкції наведених нижче задач
3. Перевірити виконання скриптів на прикладі запропонованих вхідних і вихідних даних

Задача 2.1.
Користувач має ввести два цілих числа. Програма має вивести на екран їх добуток.

Введіть перше число: 2
Введіть друге число: 3
Добуток чисел: 6

Задача 2.2.
Користувач має ввести ім’я. Програма має вивести на екран персональне привітання для користувача.

Введіть, будь ласка, Ваше ім'я
Вася
Привіт, Вася!

Задача 2.3*.
Користувач має ввести рядок і ціле додатне число. Повторити рядок вказане число разів, між рядками має бути один пробіл ‘ ’.

Введіть рядок:
дерево, а за деревом

Введіть кількість повторів:
4
дерево, а за деревом дерево, а за деревом дерево, а за деревом дерево, а за деревом

Задача 2.4*.
Користувач має ввести рядок. Програма має поміняти місцями перший і останній символ та вивести результат на екран.
Корисні функції:
`word[-1]` – останній символ

Введіть рядок
Генетика
аенетикГ

Задача 2.5.**
Користувач має ввести рядок і ціле додатне число. Якщо вказаний символ більше за довжину рядка, вивести помилку з поясненням. Інакше поміняти місцями перший і вказаний числом символ і вивести результат.
Корисні функції:
`len()` - довжина строки,
`if():, else:` - умовний оператор

Введіть рядок
Генетика
Введіть номер символа, що буде замінено
5
теневГика

Практична робота №3.

Тема: Масиви, цикли і умовні оператори Python

Перевагою комп'ютерних систем для обробки даних є потенціал до їх автоматизації. Повторювані процеси можна згрупувати в цикли і виконувати в них обчислення для кожного однотипного елементу.

Важливим компонентом обробки інформації є побудова логічних схем з розгалуженням алгоритму дій у відповідності до виконання або невиконання умов схеми.

Масиви представляють собою перелік даних в одній змінній

```
array = [1,2,3,4] # масив даних
```

Звернутись до елементу масиву можна з використанням індексу (порядкового номеру) елементу в квадратних дужках

```
print(array[2]) # 3 (індекс 2 має третій об'єкт в масиві)
```

Перевести строку в масив можна функцією split(). За замовченням роздільником є пробіл.

```
array = '1 2 3 4 5'.split() # змінній array буде
присвоєний масив ['1', '2', '3', '4', '5']
```

Створити масив послідовних цілих можна командою **range**

```
range(5) # [0, 1, 2, 3, 4]
```

Оператор **for** використовується для формування циклічних подій з елементами массиву

```
for i in range(5):
    print(i)
```

Код `print(i)` буде виконаний 5 разів, кожен раз з іншим значенням змінної.

(!) Зверніть увагу, що рядок всередині циклу пишеться з **відступом**. Відступ формується клавішею «Tab» або 4 пробілами. Все, що буде написано з відповідним відступом виконується всередині циклу, якщо

відступ прибрати, подія буде виконана за межами циклу (після його завершення).

```
for i in [1,2,3,4,5]:
    print(i)
    print(i*2)
print(i*3)
```

В прикладі вище події `print(i)` та `print(i*2)` будуть виконані по 5 разів, а подія `print(i*3)` буде виконана лише 1 раз з останнім значенням змінної

Цикли можна вкласти один в один

```
for i in [1,2,3]:
    for i in [4,5]:
        print(i*j)
```

Подія `print(i*j)` відбудеться 6 разів, кожен раз з різною комбінацією змінних `i` та `j`

Альтернативно, сформувати цикл можна оператором while – поки виконується умова оператора, буде відбуватись повторення циклу

```
i = 100
while (i > 0):
    print(i)
    i = i - 10
```

Одна з реалізацій **умовного оператора** в Python досягається командами if та else. Команди відокремлюються відступами. Синтаксис умови наступний

```
if умова:
    # Дії, якщо умова виконується
else:
    # Дії, якщо умова не виконується
# Дії по завершенню перевірки умови
```

Умови можна вкладати одна в одну. Для цього, як і в прикладі з циклами, використовуються відступи різних рівнів

```
if умова1:
    if умова2:
        # Дії, якщо умови 1 і 2 виконуються
    else:
        # Дії, якщо умова 1 виконується, а умова 2 – ні
else:
    # Дії, якщо умова 1 не виконується
```

Команда **brake** припиняє виконання найближчого (внутрішнього) циклу, в якому вона знаходиться.

```
for i in range(3):
    print('Зовнішній цикл, ітерація ',i)
        for j in range(3):
        print('Вкладений цикл, ітерація ',j)
        if j == 1:
            break  # закінчити вкладений цикл
```

Умова.

В якості умови використовуються Булеві значення. Це тип даних, який може приймати тільки 2 варіанти - «Так, або ні», (*`True False`*). Будь який тип даних, який буде записаний в якості умови буде спочатку конвертований в Булеві значення, а потім переданий в конструкцію if else (Якщо значення «Так», виконаються дії під **if**, якщо значення «Ні», виконується дії під **else**)

В якості умови можна передати будь яке значення – функцію, змінну, сталу; але найпоширеніше використання умов – це оператори порівняння:

`(a < b)` – вертає «Так», якщо a менше b;
`(a <= b)` – вертає «Так», якщо a менше, або дорівнює b;
`(a == b)` – вертає «Так», якщо a дорівнює b;
`(a != b)` – вертає «Так», якщо a не дорівнює b ;

***(!) Зверніть увагу** на відмінність символів присвоєння («=») та порівняння («==»).* Умова `if (a = b):` є синтаксичною помилкою.

Так як перед обробкою умови вона переводиться в Булеве значення, то виконанням умови вважається число, що не дорівнює 0 чи 0.0, та рядок ненульової довжини.

Умовами, де виконаються дії під **if** будуть наступні приклади:

```
if True :
if -1 :
if 0.003 :
if "0" :
if " " :
```

Умовами, де виконаються дії під **else** будуть наступні приклади:

```
if 0 :
if 0.0 :
if False :
if "" :
```

Умови можна групувати логічними операторами. Найбільш використовувані: **and** та **or**

```
True and True == True
True and False == False
True or False == True
False or False == False
```

Мета роботи: Ознайомитись з багатовимірними типами даних Python. Засвоїти навички автоматизації повторюваних процесів з використанням циклів та умовних операторів.

Матеріали та обладнання: Оточення Miniconda3, інтерпретатор Jupyter

1. Ознайомитись з документацією щодо використання циклів і умовних операторів в Python

2. В оточенні Jupiter написати скрипти, що виконують інструкції задач
3. Перевірити виконання скриптів на прикладі запропонованих вхідних і вихідних даних

Задача 3.1.
Користувач має ввести два цілих числа.
Якщо перше число менше другого, програма повинна вивести суму цих чисел.
У випадку, якщо перше число не менше другого, програма повинна вивести їх добуток.

Введіть перше число: 5
Введіть друге число: 6
Результат: 11

Задача 3.2*
Користувач має ввести два цілих числа і рядок математичної дії.
Програма перевіряє, чи введена дія в переліку доступних (+, -, *, /), та чи можлива операція.
Якщо введена недоступна дія або операція неможлива (наприклад, ділення на нуль), програма виводить помилку з поясненням.
У випадку успішної операції програма виводить результат.

Введіть перше ціле число: 5
Введіть друге ціле число: 6
Введіть математичну дію (+, -, *, /): *
Результат: 30

Задача 3.3**
Користувач має ввести рядок, в якому числа розділені пробілами, і рядок з математичними діями (+, -, *, /), також розділеними пробілами.
Програма виконує всі математичні операції послідовно над числами у введеному рядку, використовуючи вказані дії.
Якщо кількість чисел більша, ніж кількість дій, то в не задані дії будуть застосовані "+".
Якщо кількість дій більша, ніж кількість чисел, програма виведе помилку.

Якщо введена недоступна дія або операція неможлива (наприклад, ділення на нуль), програма також виведе помилку.
У випадку успішної операції програма виводить цілу частину результату.

```
Введіть рядок з числами через пробіл: 4 56 767 8 45 3 5 67 8 67 7 6
Введіть рядок з математичними діями через пробіл: / * + - - - *
Результат: 744
```

Задача 3.4
Користувач має ввести сторону квадрата у вигляді цілого числа
Вивести квадрат з символів "Ж "

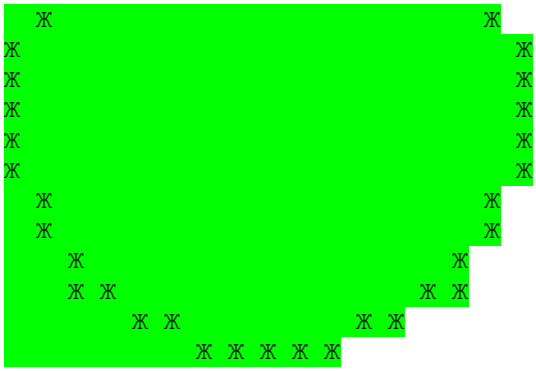
```
Введіть довжину сторони квадрата:  5
Ж Ж Ж Ж Ж
Ж Ж Ж Ж Ж
Ж Ж Ж Ж Ж
Ж Ж Ж Ж Ж
Ж Ж Ж Ж Ж
```

Задача 3.5*
Користувач має ввести ціле число, що означатиме висоту у рядках рівнобедреного трикутника
Вивести трикутник з символів "Ж "

```
Введіть висоту рівнобедреного трикутника: 9
                  Ж
               Ж Ж Ж
            Ж Ж Ж Ж Ж
         Ж Ж Ж Ж Ж Ж Ж
      Ж Ж Ж Ж Ж Ж Ж Ж Ж
   Ж Ж Ж Ж Ж Ж Ж Ж Ж Ж Ж
Ж Ж Ж Ж Ж Ж Ж Ж Ж Ж Ж Ж Ж
```

Задача 3.6**
Користувач має ввести ціле число, що означатиме радіус кола у рядках
Вивести коло з символів "Ж "

```
Введіть радіус кола: 8
            Ж Ж Ж Ж Ж
         Ж Ж         Ж Ж
      Ж Ж               Ж Ж
      Ж                   Ж
   Ж                       Ж
```


Ж Ж
Ж Ж
Ж Ж
Ж Ж
Ж Ж
Ж Ж
Ж Ж
Ж Ж
Ж Ж
Ж Ж
Ж Ж
Ж Ж Ж Ж
Ж Ж Ж Ж Ж

Практична робота №4

Тема: Модулі Python, генерація випадкових значень. Обробка винятків. Використання Python в аналізі полімерів

В біологічних системах значну роль відіграють **полімерні молекули**, які складаються з обмеженої кількості можливих мономерів. Яскравими представниками таких полімерів є нуклеїнові кислоти, що складаються з 4 варіантів 5 канонічних нуклеотидів, і білки, що до посттрансляційної модифікації є послідовністю з 20 (зрідка 21) канонічних амінокислот. Аналіз послідовностей мономерів у складі таких полімерів може дати значну інформацію про їх властивості.

Нуклеотидні послідовності у складі дволанцюгової молекули ДНК мають **відповідати певним умовам**. Ланцюги мають бути розташовані антипаралельно і у протилежних позиціях мають знаходитись комплементарні нуклеотиди. Всього можливі лише 2 комплементарні пари - аденін(A) з тиміном(T) та цитозин(C) з гуаніном(G). Також варто врахувати, що стабільність подвійної спіралі є тим більшою, чим вища частка GC-пар.

Структурування глобулярного білка зазвичай відбувається в декілька етапів. Необхідною умовою для формування глобули є наявність ділянок із регулярним чергування амінокислотних залишків по одному з двох патернів - для α-спіралі чи β-шару. Амінокислотні залишки характеризуються різною полярністю – ця ознака є ключовою при формуванні гідрофобного ядра глобулярної молекули. Отже, маючи амінокислотну послідовність, можна передбачити ймовірність утворення молекулою регулярних вторинних структур, їх відносні позиції і типи.

Мова Python в стартовому вигляді має обмежену кількість інструментів, але широкі можливості для розширення, завдяки стороннім модулям. **Модулем** називається зовнішній файл з текстом програми. Модуль може бути створений власноруч. Крім того, доступні модулі зі стандартної бібліотеки - вони вже встановлені, але не підключені.
Для **підключення модуля** використовується команда import

```
import math
print(math.pi) # 3.141592653589793
```

Для автоматизації роботи корисним інструментом може слугувати **генерація випадкових чисел**. Якщо наявна певна умова до полімеру, але невідома послідовність, яка буде відповідати умові, то продуктивним напрямком може бути генерувати випадкові послідовності, перевіряти їх відповідність умові та проводити фільтрацію звичайними чи евристичними методами.
Для генерації випадкових (псевдовипадкових, тобто криптографічно не захищених) даних використовується **модуль random**

```
import random
print(random.randint(0,5)) # випадкове ціле число від 0 до 4
```

Корисною функцією при роботі зі строками є count. Вона дозволяє порахувати, скільки разів певний рядок зустрічається у вихідному рядку

```
'11231231'.count('12') # 2
```

Конструкція **try...except** в мові програмування Python використовується для обробки винятків (exceptions) або помилок, які можуть виникнути під час виконання програми. Ця конструкція дозволяє програмі продовжувати виконання після виникнення помилки і забезпечує можливість відловлювати та обробляти ці помилки.

```
try:
    # Код, який потенційно може викликати помилку
except ExceptionType as e:
    # Код, який виконується при виникненні помилки, в змінну e при цьому буде поміщена інформація про помилку, яка виникла
```

Якщо в блоку try виникає помилка, виконання коду у блоку try переривається, і виконується блок except. Якщо помилка не виникає, блок except пропускається.

Ось приклад використання try...except для обробки помилки при діленні на нуль:

```
user_input = 0
try:
    result = 10 / user_input # Спроба ділення на нуль
except ZeroDivisionError as e:
    print("Помилка:", e)  # Виведемо повідомлення про помилку
```

Цей код не завершить виконання програми через помилку ділення на нуль, завдяки try...except, ви зможете вивести повідомлення про помилку, і програма буде продовжувати виконуватись.

Мета роботи: Ознайомитись з підключенням сторонніх модулів Python. Навчитись обробці винятків. Опанувати використання мови для аналізу біологічних процесів.

Матеріали та обладнання: Оточення Miniconda3, інтерпретатор Jupyter

Хід роботи:

1. Ознайомитись з документацію щодо модуля random
2. В оточенні Jupiter написати скрипти, що виконують інструкції задач
3. Перевірити виконання скриптів на прикладі запропонованих вхідних і вихідних даних

Задача 4.1.
Користувач має ввести рядок з нуклеотидною послідовністю у вигляді переліку однолітерних скороченнь
`Вивести цілу частину відсотоку GC`

Введіть нуклеотидну послідовність:
TGATTAACCACGTGAGCGACGTAACAGCGCCCTTTGGCAC
Відсоток GC: 55

Задача 4.2*.
Користувач має ввести ціле число в діапазоні від 50 до 70. Це число буде представляти температуру плавлення молекули ДНК в градусах Цельсія. Температура плавлення ДНК залежить від вмісту пар "GC" у

послідовності, і ми будемо враховувати, що з 0% GC-пар температура плавлення - 50°C, а з 100% GC-пар - 70°C.
Завдання полягає в наступному:

1. Згенерувати 20 випадкових послідовностей нуклеотидів (праймерів) по 30 нуклеотидів кожен.
2. Обчислити температуру плавлення кожного праймера, враховуючи вміст GC-пар в ньому.
3. Обрати той праймер, у якого температура плавлення найближча до введеного числа.
4. Вивести послідовність цього праймера та цілу частину його температури плавлення.

Введіть температуру плавлення (50-70°C): 57
Послідовність праймера: CCTAGGAGTCTTTAGTTATGTTAAGAGAAC
Температура плавлення: 57 °C

Задача 4.3*.
Користувач має ввести послідовність з 40 нуклеотидів (букв A, C, G, T). Програма повинна знайти найдовший паліндром у введеній послідовності. Паліндром - це послідовність символів, яка читається однаково зліва направо і справа наліво.
Для цієї задачі, паліндроми можуть бути типу AAGGAA або AAGTGAA.
Програма виводить найдовший паліндром у введеній послідовності.

Введіть послідовність з 40 нуклеотидів (A, C, G, T):
AGCCTACGTACGTACGTACGATGCTACGCTATAGCATGCGTAGCGTACG
Найдовший паліндром: ATGCGTA

Задача 4.4.
Завдання полягає в тому, щоб обчислити кількість β-ділянок у послідовності амінокислот. Кожна β-ділянка має відповідати певним умовам:

Мотив послідовності для β-ділянки повинен складатися з чергування гідрофільних і гідрофобних амінокислот на відстані не менше 16 амінокислот.

Гідрофільні амінокислоти: STNQKRDEH.

Гідрофобні амінокислоти: YWVILCMF.

Пролін P і гліцин G не повинні бути присутніми на ділянці. Вам потрібно обчислити кількість таких β-ділянок у введеній послідовності амінокислот і вивести це число.

Введіть послідовність амінокислот:
KTNKSGPDNQPDSLEWQYSFQWRFELRLEIHLHMHITFSYSCKLDMEMDP
EEPEQQNHTDMHYTMTFQLSIHYRWTCKWRMTTRSTTKPRRPHIDYQVEC
TFHYQCRCEYSLRWDWSYSVECDWSNKGNETRYDVQMEWSLEINWDYE
CSVECTLDWRFNFQMRLNYSEEDQNKEDSKTKKPKSDQES
Кількість β-ділянок: 4
Перелік β-ділянок:
['SLEWQYSFQWRFELRLEIHLHMHITFSYSCKLDMEMD',
'DMHYTMTFQLSIHYRWTCKWRMT',
'HIDYQVECTFHYQCRCEYSLRWDWSYSVECDWS',
'RYDVQMEWSLEINWDYECSVECTLDWRFNFQMRLNYS']

Задача 4.5*.
Завдання полягає у створенні програми, що буде генерувати послідовність амінокислот за вказаними користувачем параметрами. Від користувача потрібно отримати інформацію про максимальну довжину поліпептиду і кількість регулярних структур кожного типу.
Потрібно врахувати наступні вимоги до регулярних структур:
Гідрофільні амінокислоти: STNQKRDEH.
Гідрофобні амінокислоти: YWVILCMF.
Пролін P і гліцин G не повинні бути присутніми на ділянці регулярної структури.
Умови для бета-складки – чергування гідрофільних і гідрофобних залишків 1 через 1 на ділянці не коротше 16 амінокислот
Умови для альфа-спіралі – тандемний повтор елементу послідовності із 1-2 гідрофільних і 2 гідрофобних залишків на ділянці не коротше 12
Якщо обраної довжини не вистачить на перелічені структури, вивести помилку. Інакше, вивести послідовність однобуквених скороченнь амінокислот, яка задовільняла б кількості заданих ділянок.

Введіть максимальну довжину поліпептиду: 300
Введіть число α-спіралей: 3
Введіть число β-структур: 4

Поліпептид:

GDGRTRHGHVVTCINNYCTDVMTRDETRTRKSGDRTSWQYKCHFQWN
VDFKWRIQCTVRLRFRFHYHKTGRTGEEIYRDCLNYWSRWLRRNEPNSS
NILQDIYDLYSMVTDEPDGQKDEDPTIKFTMDYNVEIKCHMNFDLDITYQV
EDDHPQHESSPSMRMQLNLSVHVRIHWRYTLSHHQDHGSQQDKPEDG
EHNLQCRYKYNYRYNYKLDCQMSIDWSMKFTVNHGRHGDTERRPENHS
KKEPGQ

Задача 4.6.**

Завдання полягає у знаходженні потенційних ділянок регулярної структури в послідовності амінокислот.

Врахуйте наступні умови для регулярних структур

Гідрофільні амінокислоти: STNQKRDEH.

Гідрофобні амінокислоти: YWVILCMF.

Пролін P і гліцин G не повинні бути присутніми на ділянці регулярної структури.

Умови для бета-складки – чергування гідрофільних і гідрофобних залишків 1 через 1 на ділянці не коротше 16 амінокислот

Умови для альфа-спіралі – тандемний повтор елементу послідовності із 1-2 гідрофільних і 2 гідрофобних залишків на ділянці не коротше 12 амінокислот

Передбачте можливі точки виникнення і перехопіть помилки типу ZeroDivisionError

Результатом роботи програми має бути кількість регулярних структур кожного типу і їх послідовності.

Введіть послідовність амінокислот:
GDGRTRHGHVVTCINNYCTDVMTRDETRTRKSGDRTSWQYKCHFQWNVD
FKWRIQCTVRLRFRFHYHKTGRTGEEIYRDCLNYWSRWLRRNEPNSSNILQ
DIYDLYSMVTDEPDGQKDEDPTIKFTMDYNVEIKCHMNFDLDITYQVEDDHP
QHESSPSMRMQLNLSVHVRIHWRYTLSHHQDHGSQQDKPEDGEHNLQCR
YKYNYRYNYKLDCQMSIDWSMKFTVNHGRHGDTERRPENHSKKEPGQ
Число α-спіралей 3
Число β-структур: 4
α-спіралі: ['HVVTCINNYCTDVMT', 'EIYRDCLNYWSRWLR',
'NILQDIYDLYSMVT']

β-структури: ['SWQYKCHFQWNVDFKWRIQCTVRLRFRFHY', 'TIKFTMDYNVEIKCHMNFDLDITYQV', 'SMRMQLNLSVHVRIHWRYTL', 'NLQCRYKYNYRYNYKLDCQMSIDWSMKFTV']

Практична робота №5

Тема: Створення власних функцій**.** Встановлення сторонніх модулів. Модуль Biopython. Форматовані дані.

Визначення власних функцій в Python - це ключовий аспект програмування, який дозволяє вам створювати власні блоки коду для виконання конкретних завдань. Виклик власних функцій реалізується методом, подібним до виклику вбудованих функцій.

Щоб визначити функцію, потрібно скористатись комадою **def**

```
def greet(name):
    print(f"Привіт, ",name,”!”,sep=’’)
```

Після команди def іде ім’я функції і потім в круглих дужках її аргументи через кому.
Аргументи – це змінні, які використовуються в тілі функції.
Результатом роботи функції може бути виконання дій (в наведеному вище прикладі, функція реалізує вивід інформації на екран)
В такої функції немає продукту виконання, інакше кажучи, вона нічого не повертає.

```
a = greet('Остап')
```

Тобто змінна `a` матиме значення `None`**.** Такі функції називають мовчазні (void)
Для того, щоб функція повертала результат роботи, використовується команда **return**

```
def greet(name):
    return("Привіт, "+name+"!")
a = greet('Остап')
print(a)
```

В цьому випадку в змінну `a` запишеться результат роботи функції, і вона зберігатиме рядок «Привіт, Остап!»

(!)Зверніть увагу, команда **return** закінчує виконання функції.
Наприклад:

```
def greet(name):
    return("Привіт, "+name+"!")
    print("Не забудь покласти праймери в морозильну камеру")
```

```
a = greet('Остап')
print(a)
```
Буде проігнорований вивід команди print в тілі функції

Всі змінні, які використовуються в тілі функції належать лише їй, і поза межами функції до них немає доступу
```
a = 2
def add(a):
    b = 4
    a = a + b
    print ('a в тілі функції дорівнює ',a) #6
add(a)
print('a за межами функції дорівнює ',a) #2
print('b за межами функції дорівнює ',b) # NameError: name
'b' is not defined
```

В стандартно визначену функцію потрібно передавати стільки ж аргументів, скільки було вказано при її визначенні. Але є можливість вказати значення аргументу за замовченням. В цьому випадку, аргумент можна не вказувати при виклику функції.
```
a = 2
def adder(a, b=4):
    a = a + b
    print ('a в тілі функції дорівнює ',a)
adder(a) #6
adder(a,8) #12
adder(4) #8
```

Однією з важливих переваг використання мови Python є можливість застосування продуктів роботи інших розробників. Для цього з програми для розширення базових функцій створюють модулі, які можуть розповсюджуватись у вигляді файлів.
Для використання **сторонніх модулів** їх необхідно встановити в систему. Для цієї задачі існує пакетний менеджер PIP, який дозволяє встановлювати і оновлювати модулі для Python.
Використання модулів допомагає скоротити час на розробку програми і дозволяє дослідникам обмінюватись своїми програмами.

Biopython Project - це міжнародна асоціація розробників відкритих інструментів Python для розрахунків в молекулярній біології
Biopython - це набір інструментів, що дає можливість автоматизувати задачі розрахунків шляхом використання стандартизованих форматів файлів в базах даних.
Формат FASTA - текстовий формат для нуклеотидних чи амінокислотних послідовностей, в яких нуклеотиди або амінокислоти позначаються за допомогою однолітерних скорочень.
Послідовності може передувати його ідентифікатор в базі даних GenBank і короткий опис
Приклад даних у FASTA форматі:

```
>IMGA|Medtr6g023700.1 Triacylglycerol lipase 2
   chr06_pseudomolecule_IMGAG_V3.5 5390597-5393824 H
   EGN_Mt100125 20100825
MASLGSMNIVTLTFCVIILTTCNHQAHASSRVFLNKKNDKSPIQGLCASSVTIHGFK
   CEE
HEVITKDGYILSIQRIPEGRSEAKSNVTKKKEPVIVQHGVFVDGATWFLNSPKQNLP
   MIL
ANNGFDVWIPNTRGTKFSRKHTSLDPSNKTYWDWSWDELVTYEMPAIFDFISKQTGG
   QKI
HYVGHSLGTLTALASLAEGKWENQVKSVALLSPVAYLSQMKSILGQIAARSLLSKEC
   QEK
LAQSECVGATWKRKYDEAMLKMETMSGEIEQREHEVHKLRRQIVKKNVQIELRAQGY
   HNL
SAQGSVGSSSKMHIQILMNSLLQRA*
```

Модуль **Bio.Sec** призначений для представлення біологічних послідовностей за допомогою літер:

```
from Bio.Seq import Seq
my_seq =
Seq("MKQHKAMIVALIVICITAVVAALVTRKDLCEVHIRTGQTEVAVF")
print(my_seq)

from Bio.SeqUtils import gc_fraction
gc_fraction ("ACTGN") # 40.0
```

Мета роботи: Навчитись визначати власні функції. Ознайомитись з модулем Biopython, орієнтованим на біологічні дослідження. Опанувати встановлення сторонніх модулів, використання модулю Biopython для обробки масивів біологічних даних.

Матеріали та обладнання: Оточення Miniconda3, інтерпретатор Jupyter, Biopython Cookbook, NCBI

Хід роботи:

1. В залежності від операційної системи встановити PIP та Biopython
    a. **Windows**. Натиснути Windows Key + R, ввести cmd.exe, натиснути Enter.
    У вікні, що відкрилось внести команди
    curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
    python get-pip.py
    pip install biopython
    b. **MacOS/Linux.** Відкрити термінал. Внести команди
    curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
    python3 get-pip.py
    pip install biopython
2. Ознайомитись з інструментами Biopython
3. Ознайомитись з бібліотекою Bio.Sec
4. В оточенні Jupiter написати скрипти, що виконують інструкції наведених нижче задач

Задача 5.1.
Напишіть скрипт, що виконує наступні дії:
Імпортує Bio
Імпортує Seq з бібліотеки Bio.Seq
Імпортує gc_fraction з бібліотеки Bio.SeqUtils
Вносить в змінну seq послідовність
Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG")
Виводить частку пар GC в цій послідовності.

```
Введіть нуклеотидну послідовність
   :ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG
Частка GC: 0.5641025641025641
```

Задача 5.2*.
Напишіть скрипт, що виконує наступні дії:
Імпортує необхідні функції
Вносить в змінну seq послідовність
Seq("GTGTAAATGAAAAAGATGCAATCTAATGTTGACGTACTCATGGCAT
GACTTTCCCATGTGGTTCTGATGGATGTCGCTCCCATGGCA")
Отримує послідовність комплементарного ланцюга
Проводить трансляцію комплементарного ланцюга, використовуючи таблицю кодонів "The Ascidian Mitochondrial Code"
Виводить амінокислотну послідовність
Корисні функції:
seq.complement()
seq.translate(table=)
CodonTable.unambiguous_dna_by_id[13]

Введіть нуклеотидну послідовність:
GTGTAAATGAAAAAGATGCAATCTAATGTTGACGTACTCATGGCATGACT
TTCCCATGTGGTTCTGATGGATGTCGCTCCCATGGCA
Амінокислотна послідовність: HIYFFYVGLQLHEYRTEGVHQDYLQRGYR

Задача 5.3.**
Напишіть скрипт, що виконує наступні дії:
Імпортує необхідні функції
Отримує з NCBI послідовність "Mus musculus mutant p53 mRNA, complete cds"
Заносить послідовність в змінну
Замінює позиції 345 та 346 на "A" та "A"
Замінює позиції з 500 по 520 включно на невідомі нуклеотиди (“N”)
Отримує комплементарний ланцюг
Транслює комплементарний ланцюг, використовуючи таблицю "Blastocrithidia Nuclear Code"
Створює з послідовності об'єкт SeqRecord
Заповнює даними id, description, annotation (зміст будь-який)
Зберігає файл у фаста-форматі.
Корисні бібліотеки:
from Bio import SeqIO,
import requests

requests.get(fasta_url)

```
Seq('AAGGACCNGACATCCATCGCTGATGTCAATCCCCCGTGGATCGTAA
GTCCGGGA...TGA')
Modified sequence saved to modified_sequence.fasta
```

Задача 5.4.**

Оберіть послідовності одного гена в різних видах дріжджів.

Напишіть скрипт, що проводить кластерний аналіз відстані між ними

Практична робота №6

Тема: Використання мови R для структурування даних. Синтаксис, типи даних.

R є мовою програмування та середовищем для статистичного аналізу та візуалізації даних. Широко використовується в аналізі даних, біоінформатиці, машинному навчанні та інших галузях. Має велику кількість сторонніх бібліотек, що значно розширюють можливості застосування.

RStudio є інтегрованим середовищем розробки (integrated development environment, IDE) для мови R. Вона містить ряд інструментів, таких як редактор коду, вікно консолі, панель керування пакетами, вікно візуалізації даних. Ці розширення допомагають більш продуктивно взаємодіяти з мовою R і її інструментами

Додатковою перевагою RStudio є розширення R Markdown, що дозволяє комбінувати мову R з результатами обробки даних. Це дозволяє створювати звіти і презентації з результатами аналізу даних без використання сторонніх застосунків.

Вбудоване вікно допомоги. Дуже корисною функцією RStudio є вбудована система документації. Щоб побачити опис команди, функції або вбудованих даних достатньо використати знак питання перед командою, щодо якої потрібна інформація.

```
?typeof
```

Типи даних. Щоб дізнатись тип даних, можна використати команду typeof. R використовує дані наступних типів: числа, символи, рядки, булеві значення (приймають значення лише "так"(true) або "ні"(false) і перелічені нижче службові типи даних.

Тут і далі, рядки з `таким форматуванням` потрібно почергово вносити в вікно консолі RStudio і спостерігати результати виконання

Щоб більш детально розглянути різні типи чисельних та рядкових даних використайте наступні рядки та ознайомтесь з результатами:

```
typeof(1)
typeof(1.0)
typeof(1+4i)
typeof('1')
typeof('123')
?as
as.integer(1.9)
as.character(3.465)
as.character(2+1+4i)
typeof(as.integer(1.9))
typeof(as.integer('1'))
typeof(as.character(1+4i))
```

Булевими (boolean) значеннями називаються такі, які можуть приймати лише одне з двох значень - TRUE або FALSE. Скорочено в R їх можна використовувати як T та F.

```
as.logical(0)
as.logical(1)
as.integer(TRUE)
as.character(TRUE)
as.integer(5.999)
?is
is.numeric(1)
is.integer(1)
is.character('1')
is.logical(1)
is.logical(TRUE)
is.logical(T)
is.logical("TRUE")
```

Службові типи даних наступні

`NaN` (not a number) - коли результатом обчислення буде не число, хоча кодом передбачений результат у вигляді числа

`NA` (not available) - звернення до значення, якого в системі немає.

`NULL` - значення відсутності об’єкта до якого відбулося звернення

Для більш детального ознайомлення зі службовими типами даних скористайтесь наступними рядками і ознайомтесь з результатами роботи коду:

```
is.na("")
is.character("")
is.na(NA)
is.na(NULL)
is.null(NULL)
is.null(NA)
is.na(d)
is.null(d)
is.nan(0)
is.nan(NaN)
is.nan(1/0)
is.na(1/0)
is.finite(1)
is.finite(1/0)
```

Оператори R. В мові використовуються оператори трьох типів.

Оператори присвоєння, що вносять значення в змінну. Якщо змінна вже була використана, її значення буде замінене на нове.

```
a <- 10 (символ « <- » можна швидко викликати комбінацією клавіш «Alt» + «-», після введення назви змінної)
a # 10
a = 12
a # 12
```

Математичні оператори

```
a + 2 # складання
a - 3 # віднімання
a * 5 # множення
a / 5 # поділ
a ** 3 # зведення в ступінь
a ^ 3 # зведення в ступінь
a %% 3 # залишок від поділу
a %/% 3 # ціле від поділу
```

Логічні оператори. Результатом роботи таких операторів є булеве число (`TRUE` або `FALSE`). Логічні оператори можна групувати з використанням |(“або”) та &(“та”)

```
b <- 8
a < b
a > b
a > a
a >= a
a == a
t <- T
f <- F
t | f
t & f
```

Складні математичні операції.

В мову вбудовані деякі більш складні математичні операції, які виконуються функціями. Наприклад модуль та корень

```
?abs
abs(-a)
?sqrt
sqrt(25)
log(1000,10)# 3 (логарифм 1000 по основі 10)
log10(1000)
pi
sin(pi/2)
```

Мова R створена для обробки масивів даних, тому крім звичайних змінних є більш **спеціалізовані типи збереження інформації**. Такі як вектори, списки, матриці і пакети даних (dataframes).

Вектором називається серія даних, збережених у вигляді однієї змінної

```
c(a,b)
a <- c(a,b)
```

Приклади створення векторів:

```
c(1:4)
c(1,2,3,4)
c(1,3:12,15,23)
```

Доступ до елементів вектора можливий за індексами.
(!) Зверніть увагу, що на відміну від більшості мов програмування, в R перший елемент має індекс 1

```
a <- c(1,2,3,0,5,6,7,8)
a[1:6]
a[c(T,T,T,F,F,F,F,T)]
1:8 == a
a[1:8 != a]
a[1:8 == a]
a[4] <- 4
(1:8)[4]
a[1:8 != a] = (1:8)[1:8 != a]
numeric(10)
character(15)
```

Списки. Дозволяють зберігати послідовність об’єктів у впорядкованому вигляді. Елементи списку можуть бути дані різних типів

```
a <- list(1:34,100:134, T, 34, 'element', '42')
a
a[1]
```

Матриці є багатовимірними форматами збереження даних. Доступ до них можливий по індексам. В матриці використовується два індекси через кому - [рядок, стовпчик].

```
a <- matrix(1:12,nrow=3,ncol=4)
a
a[1,3]
```

Мета роботи: Ознайомитись з мовою R та її синтаксисом. Опанувати використання RStudio для обробки одновимірних та багатовимірних масивів даних.

Матеріали та обладнання: R, RStudio

Хід роботи

1. Встановити R

    a. Обрати версію R відповідно до операційної системи
    b. обрати пункт **install R for the first time**
    c. Запустити програму встановлення, виконувати її інструкції
2. Встановити RStudio
3. Ввести команди, зазначені вище. Спробувати передбачити результат їх виконання. Порівняти передбачення з результатом.
4. Написати алгоритми, що будуть виконувати наступні задачі.

Задача 6.1.
Створити вектор з 50 цілих чисел у діапазоні від одиниці до 100 командою генерації випадкових значеннь sample().
Створити новий вектор тільки з тих чисел, які більше 50
Підказки (тут і далі виконайте команди в консолі, щоб отримати підказку):

```
?sample
sample(1:2,10)
a = sample(1:2,10)
a[c(1>2,2>1)] == a[c(FALSE,TRUE)]
```

Задача 6.2*.
Створити перший вектор з трьох чисел у діапазоні 50-100.
Згенерувати другий вектор з чисел у діапазоні від 1 до 100, у кількості, що дорівнює першому елементу з першого вектора.
Перевірити кожне число другого вектора, чи воно більше за друге число першого вектора. Якщо так, збільшити число на 30.
Створити новий вектор тільки з чисел, що більші за третє число першого вектора.
Підказки:

```
a[c(FALSE,TRUE,FALSE,TRUE)] == a[c(2,4)]
```

Задача 6.3.**
Створіть вектор зі 100 чисел у діапазоні 1-100.
Своріть новий вектор, в який оберіть лише ті числа, що відхиляються від середньго не більше, ніж на 1 стандартне відхилення.
Підказки:

```
?abs
?mean
?sd
```

Практична робота №7
Тема: Використання Rstudio для обробки даних. Пакети даних, умови, цикли, фактори.

Пакет даних (dataframe) - це структура даних мовою програмування R, яка зберігає дані в форматі, схожому на електронну таблицю. Ця двовимірна таблиця має рядки та стовпчики, де кожен стовпчик може містити дані різних типів, вказаних нижче. Пакети даних, зазвичай, використовуються в задачах аналізу даних, статистики та машинного навчання.

В RStudio є **вбудовані пакети даних**, які зручно використовувати з навчальною метою. Наприклад mtcars

```
?mtcars
df <- mtcars
View(df)
str(df)
```

Для пакетів даних створені інструменти для аналізу кількості рядків та стовпчиків (колонок)

```
nrow(df)
ncol(df)
```

До кожної колонки можно отримати доступ за її ім’ям

```
df$gear
df$gear == 3
```

Можна обрати частину пакету, вказавши в квадратних дужках відповідні параметри - [умова для рядка, умова для колонки]
Умовою для колонки може бути її назва чи індекс.
Відсутність умови передбачає використання всіх колонок чи рядків.
Умова зі знаком «-» передбачає використання всіх, окрім вказаних

```
df[,]
df[df$gear == 3,]
df[df$gear == 3,'carb']
df[df$gear == 3,c('gear','carb','am','vs')]
df[, c(1, 3, 4)]
```

```
df[,-c(1, 3, 4)]
```

Маніпуляції з колонками. Можливо створити колонку, для цього потрібно написати її ім’я і присвоїти значення. Це значення буде відповідати кожному рядку у цій колонці. Якщо присвоїти вектор, він буде циклічно повторювати значення.
(!)Зверніть увагу Якщо вектор більше за кількість рядків, виникне помилка.

```
df$empty <- 0
df$empty <- c(0,1,2)
df$empty <- c(1:1000)
df$empty <- c(1:nrow(df))
```

Щоб **прибрати** колонку. можливо присвоїти їй значення NULL

```
df$empty <- NULL
```

Для заповнення значеннями колонки можна використовувати дані, що вже існують у цьому пакеті

```
df$new <- df$cyl/2
df$new
```

Також можна створювати пакети даних, що будуть містити фрагменти існуючих

```
df3 <- subset(df, gear == 3)
df4 <- subset(df, gear == 4)
```

Якщо пакети мають однакову структуру колонок, їх **рядки можна об’єднувати** з використанням команди rbind(); якщо в пакетів однакова структура рядків, можна об’єднувати їх колонки командою cbind

```
df3_4 <-  rbind(df3,df4)
df_A <- df[,c('am','vs')]
df_B <- df[,c('carb','wt')]
df_AB = cbind(df_B,df_C)
```

Умови і Цикли. Для автоматизації процесу обчислення в R використовуються умови і цикли. Підходи до їх використання подібні до таких в мові Python.

Мова R, на відміну від Python, не чутлива до відступів. Якщо ви плануєте записати умовний блок більш ніж в одну строку, вказувати початок і кінець блоку потрібно в фігурних дужках

```
if(1 > 0) print(T) else print(F)
if(1 > 0) {
  print(T)
} else {
  print(F)
}
```

В даному прикладі відступи використані з метою кращого візуального сприйняття коду, програма їх ігнорує

```
a <- 10
if(a > 0){
  print('a > 0')
} else if(a < 0){
  print('a < 0')
} else if(a == 0){
  print('a = 0')
}
```

Для спрощення запису умовного оператора є функція **ifelse**, яка приймає 3 аргумента – умову; дію, якщо умова виконується; дію, якщо умова не виконується.

```
ifelse(a > 0,'a > 0','a <= 0')
a <- c(-1,2,-3,4)
ifelse(a > 0,'a > 0','a <= 0')

for(i in c(1,2,3)) print(i)
for(i in 1:30) print(i)
for(i in 1:nrow(df)) print(df$drat[i])
```

Фактори використовуються для зберігання даних, які приймають обмежену кількість можливих значень (рівнів чи варіантів). Фактори допомагають у збереженні та обробці таких даних у структурованому форматі.

Наприклад, якщо об’єкт може приймати одне з 6 значень кольору, і такі об’єкти представлені в таблиці на тисячі рядків, то раціонально замість використання назви кольору кожен раз, скористатись переліком кольорів і вказувати лише номер кольору в переліку.
Ідея раціоналізації збереження даних таким чином була покладена в такий інструмент як вектор.

Функція unique повертає вектор унікальних значень з вихідного вектора, її можна використовувати, щоб вказати всі варіанти рівнів для вектора при перетворенні вектора на фактор. Також, вона використовується за замовчуванням, якщо другий аргумент функції factor залишити порожнім

```
colors <- c("red", "green", "blue", "red", "green")
unique(colors)
factor_colors <- factor(colors)
factor_colors
levels(factor_colors)
factor_colors <- factor(colors, labels=unique(colors))
factor_colors
df$am
df$am[3]
df$am <- factor(df$am,labels = c("automatic","manual"))
df$vs <- factor(df$vs,labels = c("V","S"))
df$am[3]
```

В R, як і в Python, можлива реєстрація власних функцій за аналогічним алгоритмом. Потрібно присвоїти змінній функцію, вказавши перелік аргументів. Після реєстрації функція зберігається за назвою, що ви можете бачити на вкладці Оточення (Environment) в RStudio. Зареєстровані функції можна викликати звичним методом.

```
multiply <- function(x,y){
  return(x*y)
}
multiply(2,3)
df$am[multiply(2,3)]
```

Обробка зовнішніх файлів. Щоб взаємодіяти з зовнішнім файлом, потрібно вказати шлях до нього. Робочим каталогом за замовчуванням є директорія “/” в Linux/MacOS та “Документи” у Windows. Якщо

розташувати файл в цій директорії, для взаємодії з ним достатньо вказати тільки назву з розширенням.

```
df <- read.csv('eukaryotes.csv')
```

Мета роботи: Ознайомитись з інтегрованим середовищем розробки RStudio. Опанувати основи роботи з пакетами даних. Провести оптимізацію збереження інформації за допомогою факторів.

Матеріали та обладнання: R, RStudio, файл eukaryotes.csv (https://bit.ly/3RzBZ4p)

Хід роботи

1. Ознайомитись з можливостями мови R та RStudio
2. Ввести команди, зазначені вище. Спробувати передбачити результат їх виконання. Порівняти передбачення з результатом.
3. Написати алгоритми, що будуть виконувати наступні задачі.

Задача 7.1.

Занести у змінну типу dataframe зміст таблиці з файлу eukaryotes.csv. Створити нову змінну типу dataframe. Зберегти в неї тільки значення назв організмів (X.Organism.Name),
та довжину геному (Size.Mb.)
якщо група(Organisms.Group) організмів «Eukaryota;Plants;Land Plants»
Підказки (виконайте команди в консолі, щоб отримати підказку):

```
df <- read.csv('eukaryotes.csv')
protists <- subset(df, Organism.Groups == "Eukaryota;Protists;Other Protists")
```

Задача 7.2*.

Використати дані з з файлу eukaryotes.csv. Згенерувати новий dataframe що містить наступні 3 колонки:
FullName - колонка, що містить дані групи та дані ім'я, розділені комою
LargeGenome - колонка, що містить "large" якщо геном більше 1000 мб та "small" якщо до 1000 мб
Expand – колонка з числовим значенням Scaffolds, якщо Level дорівнює 'Scaffold' та частку GC, якщо Level не дорівнює 'Scaffold'.
Підказки:

?paste
ifelse(vector==TRUE,vector2 <- "TRUE",FALSE)

Задача 7.3.**
Використати дані з з файлу eukaryotes.csv/
Відсортувати масив за ім'ям (X.Organism.Name).
Створити функцію, що визначатиме ковзаюче середнє, що прийматиме масив даних та розмір вікна для ковзаючого середнього
Створити додатковий вектор з ковзаючим середнім частки GC (середнє значення стовпчика «GC.» в рядках від i до i+9).
Підказки :
?factor, ?order, ?length, ?mean, ?for
vector <- c(vector,new_val)
rolling_mean <- function(x, window_size) {return(window_size))}

Практична робота №8

Тема: Використання мови R для автоматизації статистичних розрахунків та візуалізації даних. Встановлення сторонніх бібліотек.

Основні інструменти мови R розроблялись для **статистичної обробки даних**. Тому мова має велику кількість потужних команд з великою кількістю тонких налаштувань.

Найпростішим прикладом статистичної обробки пакету даних є вбудована команда попереднього перегляду властивостей. Для її використання не потрібно підключати зовнішні модулі

```
?summary
df <- mtcars
summary(df)
```

Доступні окремі розрахунки по **стандартним статистичним вимірюванням**

```
median(df$mpg) # медіана
mean(df$mpg) # середнє
range(df$mpg) # вектор, що містить мінімальне і максимальне значення
sd(df$mpg) # стандартне відхилення
```

З використанням функції aggregate можливо проводити порівняння

```
mean(df$mpg[df$am=='automatic'])
ag <- aggregate(x=df$mpg, by = list(df$am), FUN = mean)
colnames(ag)
colnames(ag) <- c('Automatic/manual','Mean MPG')
ag
```

Спеціалізація мови на статистичній обробці призвела до появи **окремого синтаксису** для розрахунків. Символ “ ~” використовується як вказівник на залежність одного параметра від іншого. Групуючи параметри, можна аналізувати різні варіанти взаємозалежностей

```
aggregate(mpg ~ am, df, mean)
aggregate(cbind(mpg, disp) ~ am + vs, df, sd)
```

Серед вбудованих методів представлені **популярні статистичні критерії**, такі як тест на критерій Шапіро-Уілка (нормальність), тест на критерій Барлетта (рівність дисперсій вибірок), тест на t-критерій Стьюдента

```
?iris
ir <- iris
ir2 <- subset(ir,Species != 'setosa')
shapiro.test(ir2$Sepal.Length) #normality
bartlett.test(Sepal.Length ~ Species, ir2) #homogeneous
t.test(Sepal.Length ~ Species, ir2)
t <- t.test(ir2$Petal.Length, ir2$Petal.Width, paired=T)
str(t)
t$p.value
```

Візуалізація даних. Базовий R не має потужних інструментів для візуалізації даних

```
hist(df$mpg)
hist(df$mpg,breaks = 20, xlab = "MPG")
boxplot(mpg ~ am, df, xlab = "Automatic / Manual")
plot(df$mpg, df$hp, xlab = 'Miles per gallon', ylab =
'Horsepowers')
```

Але мова дозволяє розширювати можливості за допомогою **використання бібліотек.**
Для використання бібліотек їх потрібно завантажити з публічних репозиторієв (сховищ). Для цього існує команда install.
Завантажувати бібліотеки потрібно тільки при першому використанні на комп’ютері.

```
install.packages('psych')
install.packages('ggplot2')
```

Для **застосування бібліотеки в проекті**, її потрібно підключити командою library
Популярною бібліотекою для розширення можливостей статистичної обробки є psych.

```
library(psych)
?psych
describe(x=df[,-c(8,9)])
```

```
describeBy(df$mpg, group=list(df$vs, df$am), mat = T,
digits = 1, fast = T)
```

Декілька бібліотек в одному проекті можуть мати назви функцій, що перекриваються. Щоб **прибрати бібліотеку з проекту,** потрібно використати функцію detach

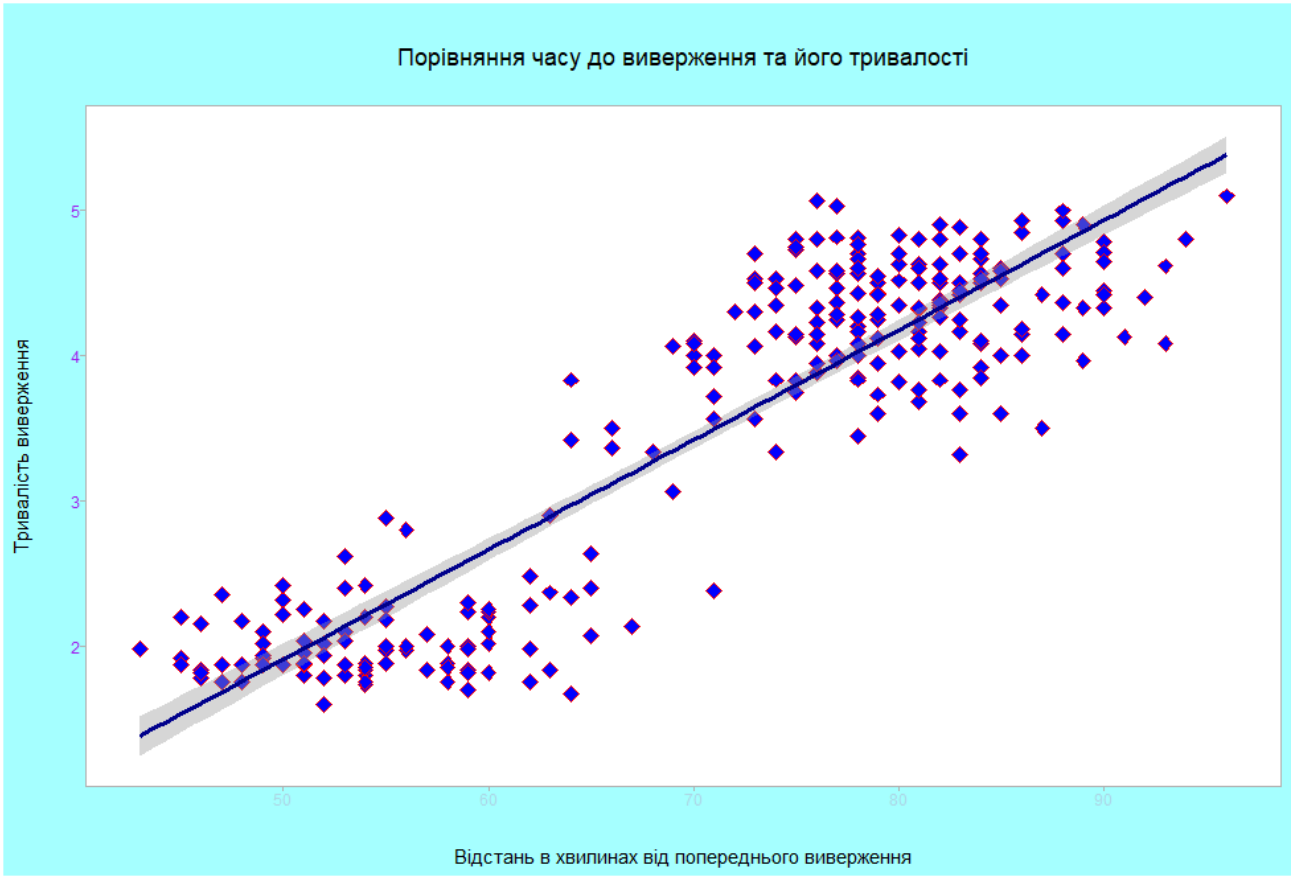
```
detach("package:psych", unload=TRUE)
```

Широко використовуваною бібліотекою для графічного представлення інформації є **ggplot2**

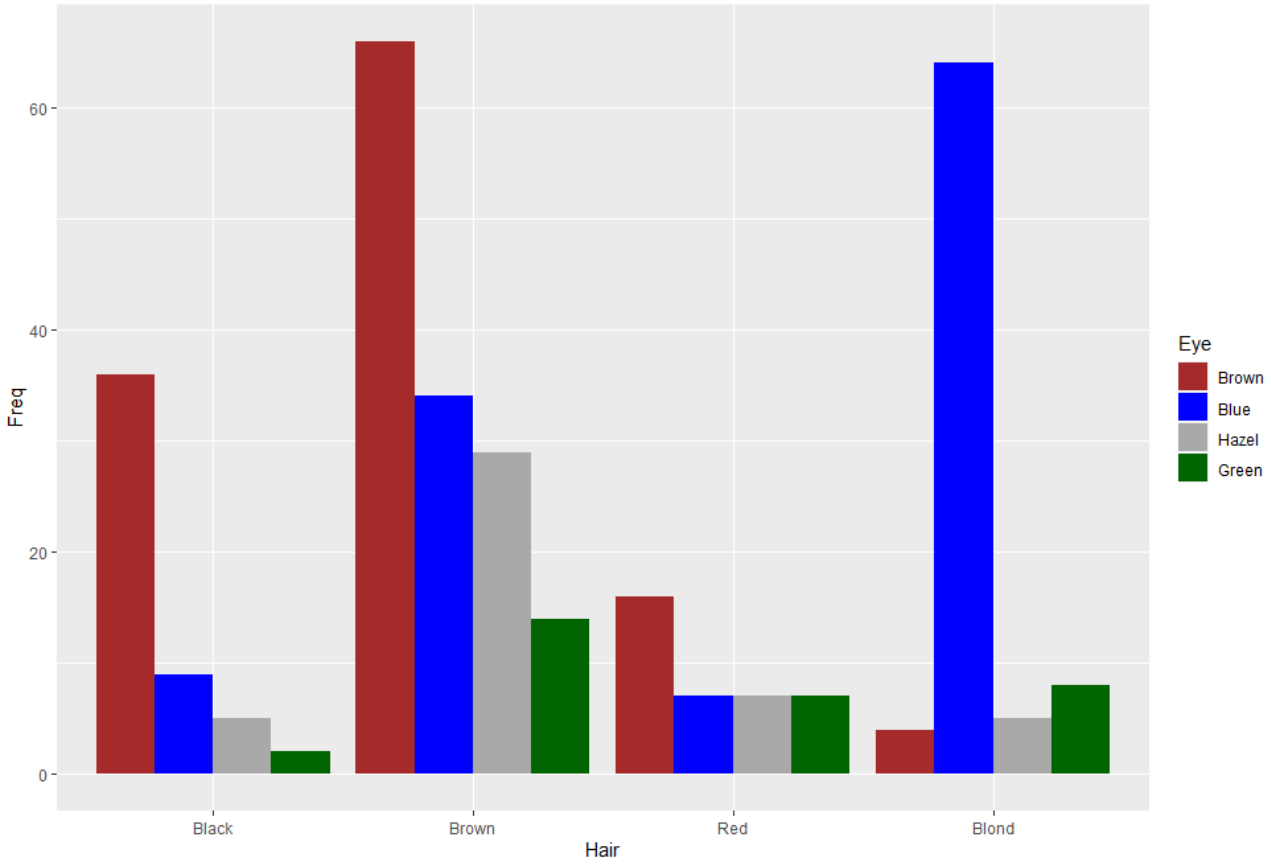
```
library(ggplot2)
?mtcars
ggplot(df, aes(x=mpg))+
  geom_histogram()
ggplot(df, aes(x=mpg))+
  geom_histogram(fill="white", col = "black", binwidth =
2)
```

За допомогою налаштувань є можливість створювати дуже деталізоване графічне представлення даних

```
data(faithful)
ggplot(faithful,aes(waiting, eruptions)) +
 geom_point(shape=23, fill="blue", color="red", size=3) +
 geom_smooth(method='lm', linewidth = 1.3, color =
 "darkblue") + xlab("Відстань в хвилинах від попереднього
 виверження") + ylab("Тривалість виверження") +
 theme_light() + theme(plot.title = element_text(hjust =
 0.5))+ theme(plot.title = element_text(hjust = 0.5,
 margin = margin(t=20, b = 20)), panel.grid =
 element_blank(), axis.text.x =
 element_text(color="lightblue", margin = margin(b =
 20)), axis.text.y = element_text(color="purple", margin
 = margin(l = 20)), plot.background = element_rect(fill =
 "#a5ffff")) + ggtitle("Порівняння часу до виверження та
 його тривалості")
```




```
mydata <- as.data.frame(HairEyeColor)
obj <- ggplot(data =
mydata[c(mydata$Sex=="Female"),c(1,2,4)], aes(x = Hair, y
= Freq,fill=Eye)) +
+     geom_bar(stat="identity", position=position_dodge())
+     scale_fill_manual(values=c("Brown", "Blue",
"Darkgrey", "Darkgreen"))
obj
```



Мета роботи: Ознайомитись з варіантами розширення функціональних можливостей проекту шляхом встановлення додаткових бібліотек. Опанувати використання базових статистичних методів мови Python. Провести порівняння базової можливості візуалізації даних і з використанням бібліотеки ggplot2

Матеріали та обладнання: R, RStudio, файл eukaryotes.csv

Хід роботи

1. Ознайомитись з можливостями бібліотеки ggplot2
2. Ввести команди, зазначені вище. Спробувати передбачити результат їх виконання. Порівняти передбачення з результатом.
3. Проаналізувати наступний алгоритм дій. Визначити, які події відбуваються в кожному рядку

```
nd <- df[,c('Level','Size.Mb.')]
nd$name <- df$X.Organism.Name
nd$sz <- nd$Size.Mb.
nd$Size.Mb. <- NULL
```

```
nd$gr <- df$Organism.Groups
nd$gc <- df$GC.
nd$l = nd$Level
nd$Level <- NULL
nd$l <- factor(nd$l, levels = unique(nd$l))
nd$name <- factor(nd$name, levels = unique(nd$name))
nd$gr <- factor(nd$gr, levels = unique(nd$gr))
df <- NULL
nd$c <- factor(nd$l!=' Chromosome' &
nd$l!='Complete',labels=c('Chromosome','Not
Chromosome'))
table(nd$c, nd$gr)
t1 <- table(nd$c, nd$gr)
prop.table(t1,2)
barplot(t1)
```

4. Написати алгоритми, що будуть виконувати наступні задачі.

Задача 8.1.

Побудуйте коробочний графік залежності кількості кінських сил авто від кількості карбюраторів у вбудованій таблиці даних mtcars.

Підпишіть вісі, додайте назву графіку

Підказки (виконайте команди в консолі, щоб отримати підказку):

```
?mtcars
boxplot(am ~ vs, df, xlab = "x",
title = "Назва",
ylab = "y")
```

Задача 8.2*.

Побудуйте точковий графік залежності миль за галон від ваги та кінських сил.

Підпишіть графік та вісі. Замініть точки на трикутники. Встановіть розмір трикутників 5.

Підказки:

```
?mtcars
install.packages('ggplot2')
library(ggplot2)
?ggplot2
?geom_point
```

```
df$a <- factor(df$a,labels = c("A","B"))
```

Задача 8.3.**
Побудуйте точковий графік залежності кінних сил від кількості циліндрів в двигуні, ваги, типу коробки передач та кількості миль за галон з таблиці даних mtcars
Підпишіть таблицю, вісі. Вкажіть легенду
Підказки:
?geom_point,
?aes,
?xlab,
?ylab,
?labs,
?ggtitle,
?factor
?shape,
?col,

Задача 8.4
Створіть dataframe з вбудованого пакету iris, в який розмістіть тільки види versicolor та setosa.
Проведіть t-тест довжини пелюсток між цими видами.
Збережіть в змінну значення p-value.
Підказки:
data(mtcars)
?t.test
?str
t$p.value

Задача 5*.
Створіть dataframe з вбудованого пакету iris, в який розмістіть тільки види versicolor та virginica.
Проведіть тест на нормальність та гомогенність вибірок.
Проведіть t-тест ширини пелюсток між видами.
Проведіть парний t-тест довжини та ширини пелюсток.
Проведіть парний t-test довжини та ширини чашолисток.
Підказки:

```
t.test(a ~ b,df)
t.test(df$a,$df$b,paired = "")
```

Задача 8.6.**

Створіть функцію aspirant_diversant, яка отримує dataframe з залежною змінною (dep) та видаляє 10 випадкових значень.

Створіть функцію phd_repair, яка знаходить відсутні значення залежної змінної (dep) в dataframe, будує лінійну регресійну модель по двум незалежним (ind_1, ind_2) та заповнює відсутні значення передбаченнями моделі.

Підказки:

```
?sample
?fit,
?lm,
?na.action,
?is.na,
?predict
```

Практична робота №9

Тема: Основи систематизації обміну даними. Типи баз даних. Основні джерела масивів даних в біології.

Велика кількість наукових груп проводить дослідження, результатом яких є **накопичення масивів даних**. Для оптимізації процесу обробки даних важливо використовувати уніфіковані підходи.

Стандартизація
Задачі уніфікації бере на себе стандартизація - це інструмент, що допомагає забезпечувати єдиний для всіх науковців підхід до вимірювання, обміну даних та інформації.
Основною метою стандартизації є створення спільної і загальноприйнятої бази термінів

Darwin Core - це стандарт, який підтримується Darwin Core Maintenance Interest Group. Він містить глосарій термінів (в інших контекстах вони можуть називатися властивостями, елементами, полями, стовпчиками, атрибутами або концепціями), призначених для полегшення обміну інформацією про біологічне різноманіття шляхом надання ідентифікаторів, міток і визначень. Darwin Core в основному базується на таксонах, їх появі в природі, задокументованій спостереженнями, зразками, пробами та пов’язаною інформацією.

Lifemap - це таксономічне дерево, що засновано на всіх існуючих наукових роботах і вважається загальноприйнятим. З великою кількістю видів пов’язані також сторінки Wikipedia, посилання на які можна побачити безпосередньо на сайті.

International Organization for Standardization ISO - це міжнародний проект, учасниками якого є національні інститути стандартизації. Вони формують спільну базу термінів, визнану всіма учасниками ISO

Для зручного обміну інформацією корисно мати уніфікований підхід до ідентифікації кожного її блоку. Цим займається некомерційна організація International DOI Foundation і її локальний підрозділ Multilingual European Registration Agency of DOI. Їх задачею є створення унікального

ідентифікаційного номеру для кожного пакету цифрової інформації згідно зі стандартами ISO (ISO 26324).

Наукометрія є невід'ємною складовою обробки наукової інформації. Вона допомагає оптимізувати вивчення джерел, формуючі рангову систему, що заснована на впливовості зазначеної інформації на науковий світ. Scientific Journal Rankings - це ресурс, що займається наукометрією на рівні наукових журналів.

Пошук даних
Під час роботи дослідницької групи важливо мати доступ до даних щодо теми дослідження, які вже були отримані в результаті наукового пошуку іншими групами.

Researchgate - мережа обміну інформацією для дослідників, що побудована на засадах соціальних мереж. Використовується для оголошення про публікації, спілкування між дослідницькими групами та швидкого обміну продуктами наукової роботи.

Пошук по статтям - спеціалізований інструмент пошукової системи Google для знаходження в базах наукових даних статей за пошуковими запитами.

Пошук по книгам - подібний до попереднього ресурс, але спеціалізується на книжках.

Web Of Knowledge - бібліотека метаданих публікацій.

Dimensions - база даних публікації з перехресними посиланнями.

Biodiversity Heritage Library - база даних сканованих версій паперових журналів, що стосуються біорізноманіття.

Агрегатори текстових наукових даних

Archive - інструмент доступу до будь-якої інформації, яка за весь час існування проекту була розміщена в мережі. Дозволяє відкривати наразі недоступні ресурси, чи попередні версії наразі змінених ресурсів.

Bioone це база даних із понад 200 видань з біологічних та екологічних наук, у відкритому доступі та за підпискою.

Jstor дослідницька та навчальна платформа, що допомагає бібліотекам підключити студентів і викладачів до важливого контенту, а видавцям охопити нову аудиторію.

MDPI - агрегатор наукових журналів з відкритим доступом. Наразі має понад 400 журналів, що працюють за ліцензією Creative Commons Attribution License (CC BY)

Спеціалізовані бази метаданих

Крім загальних агрегаторів, є спеціалізовані бази даних, де зібрана інформація конкретного типу.

ITIS - таксономічна інформація про рослини, тварин, гриби та мікроби світу

IPNI - це номенклатурний покажчик назв судинних рослин, опублікований згідно з Кодексом номенклатури водоростей, грибів і рослин (ICN).

Zoobank - онлайн реєстр зоологічної номенклатури з відкритим доступом.

CCDB - база даних номерів рослинних хромосом. CCDB має на меті об'єднати існуючі ресурси даних у велику центральну базу даних, яка регулярно оновлюватиметься спільнотою.

CanadenSys - динамічний центральний веб-портал, що надає доступ до мережевих даних про зразки (включаючи зображення та геолокацію), а також до контрольних списків з можливістю пошуку актуальних наукових назв із бази даних канадських судинних рослин (VASCAN).

IUCN Red List - червоний список видів, що перебувають під загрозою, створений Міжнародним союзом охорони природи.

Global Biodiversity Information Facility - це міжнародна мережа та дослідницька інфраструктура, що фінансується урядами світу та спрямована на надання відкритого доступу до даних про всі форми життя на Землі.

Naturalist надає місце для запису та впорядкування природних знахідок та доступ до них спеціалістам з різних галузей.

eBird Observation Dataset - підрозділ GBIF, що спеціалізується на спостереженні за птахами.

Світові бази біологічних та медичних даних

National Center for Biotechnology Information - найбільший ресурс даних в галузі біології та медицини.

European Bioinformatics Institute є частиною Європейської лабораторії молекулярної біології (EMBL), міжурядової дослідницької організації, яку фінансують понад 20 держав-членів, потенційних та асоційованих держав-членів.

Data Bank of Japan агрегатор баз даних, інструментів обробки інформації і доступу до обчислювальних можливостей інститутів.

Мета роботи: Ознайомитись з типами баз даних наукової інформації. Ознайомитись з прикладами ресурсів різноманітного призначення. Провести пошук інформації в базах даних. Ознайомитись з можливостями внесення власної інформації в світові бази даних.

Матеріали та обладнання: Доступ до мережі інтернет

Хід роботи

1. Перейти за посиланнями, переліченими вище.
2. На кожному ресурсі знайти розділ “About”, ознайомитись зі змістом розділу
3. Виконати завдання:

Завдання 9.1.
Знайти в Lifemap існуючий вид. Скопіювати посилання на Wikipedia щодо нього.

Завдання 9.2*.
На сайті GBIF створити пошук знахідок,
обвести полігоном місцезнаходження ваше місто народження,
обрати діапазон років з 1990 по 2020,
обрати набори даних лише iNaturalist Research-grade Observations,
обрати тип медіа - Зображення.

В якості відповіді на завдання внесіть дані з адресного рядка браузера після формування фільтру.

Завдання 9.3*.
Знайдіть в архіві власних фото зображення біологічного об'єкту в природному середовищі існування.
Створіть спостереження на www.inaturalist.org.
Заповніть спостереження всіми доступними даними.
В якості відповіді на завдання введіть посилання на спостереження.

Завдання 9.4.
На сайті NCBI знайти сторінку, що відповідає запиту "Canis lupus genome assembly, chromosome: 3"
Ввімкнути демонстрацію геному Display options/Show sequence => Update View.
В якості відповіді на завдання скопіюйте нуклеотиди з 1021 по 1080 цієї послідовності

Завдання 9.5.
Обрати з сайту Lifemap будь-який вид.
Знайти повногеномну послідовність або послідовність його другої хромосоми в геномних базах даних (наприклад NCBI).
Якщо геномної послідовності зазначеного виду не існує, обрати інший вид з сайту Lifemap і повторити пошук.
В якості відповіді на завдання внести посилання на сторінку обраної послідовності в базі даних. .

Варіанти рішення

```
# Python 3.9.6

#2.1
# Програма виводить на екран добуток чисел
a = int(input('Введіть перше число: '))
b = int(input('Введіть друге число: '))
print('Добуток чисел: ',a*b)

#2.2
# Програма виводить на екран персональне привітання для користувача
print('Введіть, будь ласка, Ваше ім\'я')
name = input()
print('Привіт, '+name+'!')

#2.3
# Програма повторює рядок від користувача вказану кількість разів
print('Введіть рядок')
string = input()
print('Введіть кількість повторів')
repeats = int(input())
print(repeats*(string+' '))

#2.4
# Програма міняє першу і останню літеру рядку, що буде введено користувачем
print('Введіть рядок')
string = input()
a = len(string)
print(string[a-1]+string[1:a-1]+string[0])

#2.5
# Програма міняє місцями першу і вказану літеру в рядку, що буде введено користувачем
print('Введіть рядок')
string = input()
print('Введіть номер символа, що буде замінено')
pos = int(input())
a = len(string)
```

```
if (pos > a):
    print('Помилка: Внесений номер символа відсутній в рядку')
else:
    print(string[pos-1]+ string[1:pos-1] + string[0] +
string[pos:])



#3.1
# Програма виводить суму чисел, якщо перше введене
користувачем число менше за друге і добуток в інших випадках
a = int(input('Введіть перше число: '))
b = int(input('Введіть друге число: '))
print('Результат: ', end='')
if (a < b):
    print(a+b)
else:
    print(a*b)

#3.2
# Програма виконує математичну дію з переліку (+, -, *, /) між
двома введеними користувачем числами
num1 = int(input("Введіть перше ціле число: "))
num2 = int(input("Введіть друге ціле число: "))
operation = input("Введіть математичну дію (+, -, *, /): ")
if operation not in ('+', '-', '*', '/'):
    print("Помилка: Недопустима математична дія.")
elif operation == '/' and num2 == 0:
    print("Помилка: Ділення на 0 неможливе.")
else:
    if operation == '+':
        result = num1 + num2
    elif operation == '-':
        result = num1 - num2
    elif operation == '*':
        result = num1 * num2
    else:
        result = num1 / num2
    print("Результат:", result)

#3.3
```

```python
# Програма виконує серію математичних дій з серією чисел, введених через пробіл і виводить на екран результат, округлений до цілих
numbers_str = input("Введіть рядок з числами через пробіл: ")
operations_str = input("Введіть рядок з математичними діями через пробіл: ")
numbers = numbers_str.split()
operations = operations_str.split()

if len(numbers) < len(operations):
    print("Помилка: Недостатньо чисел для виконання всіх операцій.")
else:
    operations.extend(['+'] * (len(numbers) - len(operations)))
    result = int(numbers[0])
    for i in range(len(operations)-1):
        lresult = result
        if operations[i] == '+':
            result += int(numbers[i + 1])
        elif operations[i] == '-':
            result -= int(numbers[i + 1])
        elif operations[i] == '*':
            result *= int(numbers[i + 1])
        elif operations[i] == '/':
            if int(numbers[i + 1]) == 0:
                print("Помилка: Ділення на 0 неможливе.")
                break
            else:
                result /= int(numbers[i + 1])
        else:
            print(f"Помилка: Недопустима математична дія '{operations[i]}'")
            break
        print(lresult, operations[i], numbers[i + 1] , '=', result)
    else:
        print("Результат:", round(result))

#3.4
# Програма виводить на екран квадрат з літер "Ж", вказаного користувачем розміру
```

```python
side_length = int(input("Введіть довжину сторони квадрата: "))
for i in range(side_length):
    for j in range(side_length):
        print("Ж ", end="")
    print()

#3.5
# Програма виводить на екран рівнобедрений трикутник з літер "Ж", вказаної користувачем висоти
height = int(input("Введіть висоту рівнобедреного трикутника: "))
for i in range(1, height + 1):
    for j in range(height - i):
        print("  ", end="")
    for k in range(2 * i - 1):
        print("Ж ", end="")
    print()

#3.6
# Програма виводить на екран коло, вказаного користувачем радіусу
radius = int(input("Введіть радіус кола: "))
for i in range(-radius, radius + 1):
    for j in range(-radius, radius + 1):
        distance = (i ** 2 + j ** 2) ** 0.5
        if radius - 0.5 <= distance <= radius + 0.5:
            print("Ж ", end="")
        else:
            print("  ", end="")
    print()



#4.1
# Програма визначає відсоток GC-пар введеної користувачем послідовності
sequence = input("Введіть нуклеотидну послідовність: ")
gc_count = sequence.count("G") + sequence.count("C")
total_length = len(sequence)
gc_percentage = (gc_count / total_length) * 100
print("Відсоток GC:", int(gc_percentage))
```

```python
#4.2
# Програма генерує набори праймерів з випадкових значень і виводить на екран той, що має найближчу до введеної користувачем температури
import random
MIN = 50
MAX = 70
number = 20
nucleotides = 30
while True:
    try:
        temperature = int(input(f'Введіть температуру плавлення ({MIN}-{MAX}°C): '))
        if MIN <= temperature <= MAX:
            break
        else:
            print(f'Температура повинна бути в діапазоні від {MIN} до {MAX} градусів Цельсія.')
    except ValueError:
        print("Введіть коректне ціле число.")
primers = []
for _ in range(number):
    primer = "".join(random.choice("ACGT") for _ in range(nucleotides))
    primers.append(primer)
closest_primer = None
closest_temperature = None
for primer in primers:
    temp = MIN + (MAX-MIN)*((primer.count("G") + primer.count("C")) / len(primer) )
    if closest_primer is None or abs(temp - temperature) < abs(closest_temperature - temperature):
        closest_primer = primer
        closest_temperature = temp
print("Послідовність праймера:", closest_primer)
print("Температура плавлення:", int(closest_temperature), "°C")

#4.3
# Програма визначає найдовший паліндром, що наявний в послідовності
```

```
sequence = input("Введіть послідовність з 40 нуклеотидів (A, C, G, T): ")
longest_palindrome = ""
for i in range(len(sequence)):
    for j in range(i + 2, len(sequence) + 1):
        substring = sequence[i:j]
        if substring == substring[::-1] and len(substring) > len(longest_palindrome):
            longest_palindrome = substring
print("Найдовший паліндром:", longest_palindrome)

#4.4
# Програма визначає кількість і послідовності бета-структур у введеній користувачем амінокислотній послідовності
sequence = 'g'+input("Введіть послідовність амінокислот: ")+'g'

hydrophilic = "STNQKRDEHstnqkrdeh"
hydrophobic = "YWVILCMFywvilcmf"
min_distance = 16
beta_strands = 0
seqv = []
i = 0
while i < len(sequence):
    if sequence[i] in hydrophilic:
        phobic = True
        distance = 0
        exit_for_loop = False
        for j in range(i + 1, len(sequence)):
            if sequence[j] in (hydrophobic if phobic else hydrophilic):
                phobic = not phobic
                distance += 1
            else:
                if distance >= min_distance:
                    beta_strands += 1
                    seqv.append(sequence[i:j])
                    i += distance
                    exit_for_loop = True
                else:
                    i += 1
            if exit_for_loop:
```

```
                break
    else: i += 1
print(f"Кількість β-ділянок: {beta_strands}")
print(f"Перелік β-ділянок: {seqv}")

#4.5
# Програма генерує амінокислотну послідовність за вказаними користувачем вимогами до кількості регулярних структур
import random
import sys

length = int(input("Введіть максимальну довжину поліпептиду: "))
alpha_count = int(input("Введіть число α-спіралей: "))
beta_count = int(input("Введіть число β-структур: "))
sequence = ''
b_distance = 16
a_distance = 12
hydrophilic = "STNQKRDEH"
hydrophobic = "YWVILCMF"
linker = "STNQKRDEHPG"

link = length - b_distance * beta_count - a_distance * alpha_count - 4 * (alpha_count + beta_count + 1)
if link < 0:
    print("Довжини послідовності не вистачає, щоб задовольнити умовам")
    sys.exit(1)

l = int((link / (alpha_count + beta_count + 1)) - 4 / 2)

for i in range(random.randint(4, 4 + l)):
    sequence += random.choice(linker)
while alpha_count + beta_count > 0:
    str = ''
    start = len(sequence)
    if random.randint(0, 1) == 0 and alpha_count > 0:
        print('a')
        sequence += random.choice(hydrophilic)
        k = 0
        while k < a_distance - 1 + random.randint(0, l):
            if sequence[-1] in hydrophilic:
```

```
                for j in range(2):
                    sequence += random.choice(hydrophobic)
                    k += 1
            else:
                for j in range(random.randint(1, 2)):
                    sequence += random.choice(hydrophilic)
                    k += 1
        alpha_count -= 1
        for i in range(random.randint(4, 4 + l)):
            sequence += random.choice(linker)
    else:
        print('b')
        sequence += random.choice(hydrophilic)
        for i in range(b_distance - 1 + random.randint(0, l)):
            if sequence[-1] in hydrophilic:
                sequence += random.choice(hydrophobic)
            else:
                sequence += random.choice(hydrophilic)
        beta_count -= 1
        for i in range(random.randint(4, 4 + l)):
            sequence += random.choice(linker)
    stop = len(sequence)
    print(sequence[start:stop])
print("Поліпептид: ",sequence)

#4.6
# Програма визначає кількість і послідовності бета-структур та
альфа-спіралей у введеній користувачем амінокислотній
послідовності
sequence = 'g' + input("Введіть послідовність амінокислот: ")
+ 'g'
hydrophilic = "STNQKRDEHstnqkrdeh"
hydrophobic = "YWVILCMFywvilcmf"
structure = "YWVILCMFywvilcmfSTNQKRDEHstnqkrdeh"
b_distance = 16
a_distance = 12
b_strands = 0
a_strands = 0
seqv = {'alfa': [], 'beta': []}
i = 0
while i < len(sequence) - min(b_distance, a_distance):
```

```
    if sequence[i] in hydrophilic and sequence[i + 1] in
hydrophobic:
        if sequence[i + 2] in hydrophilic:
            a_path = False
            distance = 2
        elif sequence[i + 2] in hydrophobic:
            a_path = True
            distance = 2
    else:
        i += 1
        continue
    j = i + 3
    close = False
    while j < len(sequence):
        if a_path and (
            (sequence[j - 2] in hydrophilic and sequence[j -
1] in hydrophilic and sequence[j] in hydrophilic)
            or (sequence[j - 2] in hydrophilic and sequence[j
- 1] in hydrophobic and sequence[j] in hydrophilic)
            or (sequence[j - 2] in hydrophobic and sequence[j
- 1] in hydrophobic and sequence[j] in hydrophobic)
            or (sequence[j] not in structure)
        ):
            close = True
        elif not a_path and (
            (sequence[j - 1] in hydrophilic and sequence[j] in
hydrophilic)
            or (sequence[j - 1] in hydrophobic and sequence[j]
in hydrophobic)
            or (sequence[j] not in structure)
        ):
            close = True
        else:
            distance += 1
        j += 1
        if close:
            if a_path:
                min_distance = a_distance
            else:
                min_distance = b_distance
            if distance >= min_distance-5:
                if a_path:
```

```
                    a_strands += 1
                    seqv['alfa'].append(sequence[i:i +
distance])
                else:
                    b_strands += 1
                    seqv['beta'].append(sequence[i:i +
distance])

                i += distance
            else:
                i += 1
            j+=len(sequence)
print("Число α-спіралей", a_strands)
print("Число β-структур:", b_strands)
print("α-спіралі:",  seqv['alfa'])
print("β-структури:", seqv['beta'])



#5.1
# Програма визначає частку GC-пар у введеній користувачем
нуклеотидній послідовності
import Bio
from Bio.Seq import Seq
from Bio.SeqUtils import gc_fraction
q = Seq(input('Введіть нуклеотидну послідовність :'))
print('Частка GC:',gc_fraction(q))

#5.2
# Програма симулює результат трансляції нуклеотидної
послідовності в амінокислотну і виводить останню на екран
from Bio.Seq import Seq
from Bio.Data import CodonTable

sequence = input('Введіть нуклеотидну послідовність: ')
seq = Seq(sequence)
complementary_seq = seq.complement()
ascidian_mitochondrial_table =
CodonTable.unambiguous_dna_by_id[13]
translated_seq =
complementary_seq.translate(table=ascidian_mitochondrial_table
)
```

```
print("Амінокислотна послідовність:", translated_seq)

#5.3
# Програма отримує послідовність з бази даних NCBI, вносить модифікації і зберігає в FASTA файл
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
import requests
from Bio import SeqIO
import tempfile

fasta_url =
"https://www.ncbi.nlm.nih.gov/sviewer/viewer.fcgi?id=AB021961.
1&db=nuccore&report=fasta&retmode=text&withmarkup=on&tool=port
al&log$=seqview&maxdownloadsize=1000000"
modifications = [
    {"start": 344, "finish": 346, "mod": "A" * 2},
    {"start": 499, "finish": 520, "mod": "N" * 21}
]

response = requests.get(fasta_url)
if response.status_code == 200:
    # Create a temporary file to store the fetched sequence
    with tempfile.NamedTemporaryFile(mode="w", delete=False)
as temp_file:
        temp_file.write(response.text)
        temp_file_name = temp_file.name

    sequence_record = SeqIO.read(temp_file_name, "fasta")
    for md in modifications:
        start = md["start"]
        finish = md["finish"]
        mod = md["mod"]
        sequence_record.seq = sequence_record.seq[:start-1] +
Seq(mod) + sequence_record.seq[finish:]

    complementary_sequence = sequence_record.seq.complement()
    new_seq_record = SeqRecord(complementary_sequence)
    new_seq_record.id = "Modified_Sequence"
    new_seq_record.name = "Modified_P53_Sequence"
    new_seq_record.description = "Modified sequence from NCBI"
    output_file = "modified_sequence.fasta"
```

```
    SeqIO.write(new_seq_record, output_file, "fasta")
    print(new_seq_record)
    print(f"Modified sequence saved to {output_file}")
else:
    print("Failed to fetch the FASTA sequence from the URL.")

#5.4
# Програма отримує послідовності з бази даних NCBI, проводить кластерний аналіз та виводить на екран філогенетичне дерево
from Bio import Entrez, SeqIO
from Bio.Phylo.TreeConstruction import DistanceCalculator,
DistanceTreeConstructor
from Bio.Phylo import draw_ascii
from Bio.Align import MultipleSeqAlignment

email = "sample@gmail.com"  # Вкажіть свою пошту

accessions = ["XM_046078380.1", "XM_033910137.1",
"XM_056179703.1"]
records = []

for accession in accessions:
    Entrez.email = email
    handle = Entrez.efetch(db="nucleotide", id=accession,
rettype="gb", retmode="text")
    record = SeqIO.read(handle, "genbank")
    records.append(record)
alignment = MultipleSeqAlignment(records)

calculator = DistanceCalculator('identity')
dm = calculator.get_distance(alignment)

constructor = DistanceTreeConstructor()
tree = constructor.upgma(dm)

print("Філогенетичне дерево:")
draw_ascii(tree)
```

```r
# R 4.2.2

#6.1
#Програма створює вектор з 50 чисел в діапазоні 1-100 та створює новий вектор лише з тих чисел, що більше 50
random_vector <- sample(1:100, 50, replace = TRUE)
new_vector <- subset(random_vector, random_vector > 50)
print(new_vector)

#6.2
# Програма створює вектор з 3 чисел і використовує ці числа для генерації і модифікації другого вектору
first_vector <- sample(50:100, 3)
second_vector <- sample(1:100, first_vector[1], replace = TRUE)
second_vector <- ifelse(second_vector > first_vector[2], second_vector + 30, second_vector)
new_vector <- second_vector[second_vector > first_vector[3]]
print(new_vector)

#6.3
#Програма створює вектор зі 100 чисел у діапазоні 1-100 та генерує новий вектор, лише з тих чисел, що відхиляються від середнього не більше ніж на 1 стандартне відхилення
random_vector <- sample(1:100, 100, replace = TRUE)
mean_value <- mean(random_vector)
std_dev <- sd(random_vector)
print(paste("Середнє значення:", mean_value))
print(paste("Стандартне відхилення:", std_dev))
new_vector <- random_vector[abs(random_vector - mean_value) <= std_dev]
print(new_vector)

#7.1
#Програма використовує файл eukaryotes.csv і створює новий пакет даних з вибірковою частиною даних файлу.
df <- read.csv("eukaryotes.csv", header = TRUE)
el <- subset(df, Organism.Groups == "Eukaryota;Plants;Land Plants")[,c('X.Organism.Name','Size.Mb.')]
```

```r
#7.2
#Програма використовує файл eukaryotes.csv і створює пакет
даних з частиною даних файлу, що відповідають наборам умов а
також результатами обробки даних.
df <- read.csv("eukaryotes.csv", header = TRUE)
df$LargeGenome <- ifelse(df$Size.Mb > 1000, "large", "small")
df$Expand <- ifelse(df$Level == "Scaffold", df$Scaffolds,
df$GC.)
df$FullName <- paste(df$Organism.Groups, df$X.Organism.Name,
sep = ", ")
result_df <- df[, c("FullName", "LargeGenome", "Expand")]
View(result_df)

#7.3
#Програма використовує файл eukaryotes.csv, сортує дані за
ім'ям та визначає ковзаюче середнє одного зі стовпчиків
dfo <- df[order(df$X.Organism.Name),]
rolling_mean <- function(x, window_size) {
  result <- numeric(length(x))
  for (i in 1:length(x)) {
    start <- max(1, i - window_size + 1)
    end <- i
    result[i] <- mean(x[start:end], na.rm = TRUE)
  }
  return(result)
}
window_size <- 10
gc_rolling_mean <- rolling_mean(dfo$GC., window_size)
dfo$Rolling_GCMean <- gc_rolling_mean


#8.1
# Програма використовує вбудований пакет даних mtcars і
створює коробочний графік по даним з неї
boxplot(mtcars$hp ~ as.factor(mtcars$carb),
        xlab = "Кількість карбюраторів",
        ylab = "Кількість кінських сил авто",)

#8.2
# Програма будує точковий графік з користувачськими
налаштуваннями візуальної складової
# install.packages("ggplot2")
```

```r
library(ggplot2)
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(shape = 17, size = 5) +
  labs(title = "Залежність миль за галон від ваги та кінських
сил",
       x = "Вага (Weight)",
       y = "Милі за галон (Miles per Gallon)") +
  theme_minimal()

#8.3
# Програма будує точковий графік залежності одного параметру
від набору інших
# install.packages("ggplot2")
# library(ggplot2)
ggplot(mtcars, aes(x = cyl, y = hp, color = gear, size = mpg))
+
  geom_point() +
  labs(title = "Залежність кінних сил від набору параметрів
авто",
       x = "Кількість циліндрів (cyl)",
       y = "Кількість кінних сил (hp)",
       size = "Милі за галон (mpg)") +
  scale_size_continuous(range = c(3, 10)) +
  scale_color_continuous(name = "Тип коробки передач (gear)")
+
  theme_minimal()

#8.4
#Програма використовує вбудований пакет iris, проводить
порівняння довжини пелюсток між двома видами
data(iris)
df <- iris[iris$Species %in% c("versicolor", "setosa"), ]
t_test_result <- t.test(df$Petal.Length ~ df$Species)
p_value <- t_test_result$p.value
print(p_value)

#8.5
#Програма проводить ряд статистичних аналізів з даними
вбудованого пакету iris
#install.packages("car")
library(car)
#data(iris)
```

```r
df <- iris[iris$Species %in% c("versicolor", "virginica"), ]
shapiro_test_width <- shapiro.test(df$Petal.Width)
levene_test_width <- leveneTest(Petal.Width ~ Species, data =
df)
t_test_width <- t.test(df$Petal.Width ~ df$Species)
t_test_length_width_petals <- t.test(df$Petal.Length,
df$Petal.Width, paired = TRUE)
t_test_length_width_sepals <- t.test(df$Sepal.Length,
df$Sepal.Width, paired = TRUE)
cat("Тест Шапіро-Вілка на нормальність ширини пелюсток:\n")
print(shapiro_test_width)
cat("Тест Лівена на однорідність дисперсій ширини
пелюсток:\n")
print(levene_test_width)
cat("t-Тест для ширини пелюсток між видами:\n")
print(t_test_width)
cat("Парний t-Тест для довжини та ширини пелюсток:\n")
print(t_test_length_width_petals)
cat("Парний t-Тест для довжини та ширини чашолисток:\n")
print(t_test_length_width_sepals)

#8.6
# Реєстрація двох функцій з діями видалення значенння та
відновлення значенння логістичною регресією
aspirant_diversant <- function(dataframe, dep) {
  random_rows <- sample(1:nrow(dataframe), 10)
  dataframe[random_rows, dep] <- NA
  return(dataframe)
}

phd_repair <- function(dataframe, dep, ind_1, ind_2) {
  missing_values <- dataframe[is.na(dataframe[[dep]]), ]
  observed_values <- dataframe[!is.na(dataframe[[dep]]), ]
  model <- lm(formula(paste(dep, "~", ind_1, "+", ind_2)),
data = observed_values)
  predicted_values <- predict(model, newdata = missing_values)
  missing_values[[dep]] <- predicted_values
  repaired_dataframe <- rbind(observed_values, missing_values)
  return(repaired_dataframe)
}
# Перевіримо функції
data(mtcars)
```

```
df <- mtcars
df_without_10 <- aspirant_diversant(df, "mpg")
df_repaired <- phd_repair(df_without_10, "mpg", "cyl", "hp")
```

Рекомендована література і інтерактивні ресурси

- **Путівник мовою програмування Python**
  - Автор: Олександр Мізюк
  - Посилання: https://pythonguide.rozh2sch.org.ua/
- **Python Інтро**
  - Автор: W3schoolsUA. українською
  - Посилання: https://w3schoolsua.github.io/python/python_intro.html
- **Основи програмування.Python.**
  - Автор: А.В. Яковенко
  - Посилання: https://ela.kpi.ua/bitstream/123456789/25111/1/Python.pdf
- **Інтерактивний зошит**
  - Автори: Коршун С. Б., Мунька С. В.
  - Посилання: https://sites.google.com/view/pybook
- **Математичні обчислення засобами пакету R - програмування**
  - Автор: Кальченко В. В.
  - Посилання: https://bit.ly/3RED2QF
- **Використання можливостей мови програмування та середовища R у психологічних дослідженнях: т'юторіал з базових методів.**
  - Автор: Олександр Віноградов
  - Посилання: https://bit.ly/4676GlY

Додаткова література і ресурси

- **The Python Tutorial (англійською)**
  - Автор: Python.org
  - Посилання: https://docs.python.org/3/tutorial/index.html
- **Interactive Python Tutorials (англійською)**
  - Автор: Learnpython.org
  - Посилання: https://www.learnpython.org/
- **Python Practice Online (англійською, ігрова форма)**
  - Автор: CheckiO
  - Посилання: https://py.checkio.org/