PYTHON PROWESS EMPOWER YOUR JOURNEY WITH 100 PROMPTS GPT-4 GENERATED PROGRAMMING

براعة لغة البايثون
قم بتمكين رحلتك من خلال 100
وصفة برمجية تم إنشاؤها بواسطة
GPT4

أعداد وتأليف
أستاذ
فلاح كاطع الخفاجي
مطور تطبيقات الذكاء الاصطناعي

2024

https://iraqprogrammer.wordpress.com/

بسمـــــه تعالى

براعة لغة البايثون

قم بتمكين رحلتك من خلال 100 وصفة برمجية تم إنشاؤها بواسطة

GPT4

اعداد وتاليف

أستاذ

فلاح الخفاجي

مطور تطبيقات الذكاء الاصطناعي

الاهداء

إلى والدي ووالدتي الأجلاء رحمة الله عليهما.

إلى زوجتي رفيقة الكفاح التي لم تبخل بوقت أو جهد لمساعدتي.

الى أولادي الذين كانوا معي في مسيرتي العلمية والذين تحملوا منـي كل ذلك الانشغال.

إلى جموع الأهل والأصدقاء أهدي إليكم هذا الكتاب الرقمي والعمل المتواضع وهو مجانا وهو ثمرة مجهود اكثر من سنة في التاليف والاعداد والتنظيم

شكرا لهم جميعا مع خالص تحياتي لهم.

"

إنني جاهلٌ لا أعرف إلا حقيقة واحدة،

وهي أنني لا أعرف شيئا .

"

العالم
أسحق نيوتن

المحتويات

المقدمة.

مرحبًا بك في " Python Prowess : انطلق في رحلتك البرمجية مع 100 تحدي تم إنشاؤه بواسطة GPT-4"!

في العصر الرقمي الحالي، أصبح تعلم البرمجة مهارة لا تقدر بثمن وتفتح الأبواب أمام فرص لا حصر لها. تعتبر لغة بايثون، ببساطتها وتعدد استخداماتها، إحدى لغات البرمجة الأكثر شعبية للمبتدئين والخبراء على حدٍ سواء. سواء كنت تطمح إلى تطوير تطبيقات الويب، أو تحليل البيانات، أو أتمتة المهام، فإن إتقان لغة Python يعد خطوة أساسية نحو تحقيق أهدافك.
تم تصميم هذا الكتاب ليكون رفيقك في رحلتك المثيرة لتعلم برمجة بايثون. بفضل قوة نموذج GPT-4 الخاص بـ OpenAI، نقدم لك مجموعة منسقة مكونة من 100 تحدي برمجي تم تصميمها لتعزيز مهاراتك وتعميق فهمك لـ Python. يتم إنشاء كل تحدي بدقة لتوفير مجموعة متنوعة من التمارين، التي تغطي المفاهيم الأساسية والتطبيقات العملية والتقنيات المتقدمة.

تعليمات للمبتدئين.

قبل الغوص في التحديات، دعونا نضع بعض الإرشادات لضمان تجربة تعليمية سلسة ومثمرة:

1. قم بإعداد بيئة التطوير الخاصة بك: ابدأ بتثبيت Python على جهاز الكمبيوتر الخاص بك. يمكنك تنزيل أحدث إصدار من موقع Python الرسمي (python.org) واتباع تعليمات التثبيت المتوفرة. بالإضافة إلى ذلك، فكر في استخدام بيئة التطوير المتكاملة (IDE) مثل PyCharm أو Visual Studio Code أو Jupyter Notebook لكتابة تعليمات Python البرمجية وتشغيلها بكفاءة.

2. تعرف على أساسيات بايثون: إذا كنت جديدًا في برمجة بايثون، فخصص بعض الوقت للتعرف على الأساسيات. فهم مفاهيم مثل المتغيرات وأنواع البيانات وعوامل التشغيل والعبارات الشرطية والحلقات والوظائف ومعالجة الأخطاء. يمكنك العثور على الكثير من الموارد والبرامج التعليمية والدورات التدريبية المناسبة للمبتدئين عبر الإنترنت لمساعدتك على فهم هذه الأساسيات.

3. تعامل مع كل تحدٍ بفضول ومثابرة: بينما تتعامل مع التحديات الواردة في هذا الكتاب، احتضن عقلية النمو. لا تثبط من النكسات أو الصعوبات. بدلًا من ذلك، انظر إلى كل تحدٍ باعتباره فرصة للتعلم والتحسين. قم بتجربة أساليب مختلفة، واطلب المساعدة عند الحاجة، واحتفل بالتقدم الذي أحرزته على طول الطريق.

4. اقرأ بيان المشكلة بعناية: قبل البدء في البرمجة، اقرأ بيان المشكلة بعناية لكل تحدي. فهم المتطلبات والقيود والمخرجات المتوقعة. قم بتدوين أي نماذج من المدخلات والمخرجات المقدمة، لأنها يمكن أن ترشدك في صياغة الحل الخاص بك.

5. قم بتقسيم المشكلة: قم بتقسيم المشكلة إلى مهام أصغر يمكن التحكم فيها. تحديد الخطوات الأساسية اللازمة لحل المشكلة وتحديد خطة العمل. سيساعدك هذا النهج على مواجهة التحديات المعقدة بشكل أكثر فعالية وتجنب الشعور بالإرهاق.

6. اختبر الكود الخاص بك: بعد كتابة الحل الخاص بك، اختبره جيدًا باستخدام مدخلات مختلفة للتأكد من صحته وقوته. انتبه إلى حالات الحافة والأخطاء المحتملة التي قد تنشأ. قم بتصحيح أي مشكلات تنشأ وقم بتحسين الحل الخاص بك حسب الحاجة.

7. تأمل وتعلم: بمجرد الانتهاء من التحدي، خذ لحظة للتفكير في أسلوبك والدروس المستفادة. فكر في حلول بديلة، وقم بتحليل كفاءة التعليمات البرمجية الخاصة بك، وحدد مجالات التحسين. ستعمل هذه الممارسة التأملية على تعميق فهمك لبرمجة بايثون وتعزيز مهاراتك في حل المشكلات.
مع وضع هذه التعليمات في الاعتبار، تصبح جاهزًا للشروع في رحلة تعلم البايثون باستخدام منصة الدردشة واداة . GPT4.

لقد حاولت قدر المستطاع ان اخرج بأعداد وتاليف هذا الكتاب باتقان عالي ومجرب لاغلب الوصفات ومع هذا يبقى عملا بشرياً وايضا بمساعدة ادوات الدردشة المذكورة في عنوان الكتاب يحتمل النقص، فاذا كان لديك أي ملاحظات فلا تتردد حول هذا الكتاب،بمراسلتنا عبر بريدنا الالكتروني falahgs07@gmail.com

او في روابط مواقعي المذكورة في نهاية الكتاب.

دعونا تبدأ الرحلة!

100 Prompts Code Python for beginners

100 مطالبة كود بايثون للمبتدئين

في هذا القسم تم ترتيب الموجهات النصية من الوصف البسيط الى الوصف المعقد قبل الغوض برمجيا وعمليا وباستخدام واجهة GPT4 or GPT3.5 المستخدمة في استخدام الوصفات المذكوره والمرتبة ادناه ...

1. Write a Python program to print "Hello, World!"
2. Write a Python program to add two numbers.
3. Write a Python program to calculate the area of a triangle.
4. Write a Python program to swap two variables.
5. Write a Python program to generate a random number.
6. Write a Python program to convert kilometers to miles.
7. Write a Python program to check if a number is even or odd.
8. Write a Python program to check if a year is a leap year or not.
9. Write a Python program to solve a quadratic equation.
10. Write a Python program to calculate the factorial of a number.
11. Write a Python program to create a list and append items to it.
12. Write a Python program to sort a list in ascending order.
13. Write a Python program to sort a list in descending order.
14. Write a Python program to remove duplicates from a list.
15. Write a Python program to count the number of elements in a list.
16. Write a Python program to find the maximum and minimum number in a list.
17. Write a Python program to create a tuple and access its elements.
18. Write a Python program to convert a tuple to a list.
19. Write a Python program to create a dictionary and access its elements.
20. Write a Python program to update the value of a key in a dictionary.
21. Write a Python program to remove a key from a dictionary.
22. Write a Python program to sort a dictionary by key.
23. Write a Python program to sort a dictionary by value.

24. Write a Python program to create a set and add items to it.
25. Write a Python program to remove an item from a set.
26. Write a Python program to perform set operations like union, intersection, difference, and symmetric difference.
27. Write a Python program to create a function and call it.
28. Write a Python program to create a function with arguments and return a value.
29. Write a Python program to create a lambda function.
30. Write a Python program to create a class and an object.
31. Write a Python program to create a class with a method and call it.
32. Write a Python program to create a class with a constructor.
33. Write a Python program to create a class with inheritance.
34. Write a Python program to handle an exception using try-except block.
35. Write a Python program to handle multiple exceptions.
37. Write a Python program to create a file and write to it.
38. Write a Python program to read a file.
39. Write a Python program to append text to a file.
40. Write a Python program to delete a file.
41. Write a Python program to create a module and import it.
42. Write a Python program to install a package using pip.
43. Write a Python program to create a virtual environment.
44. Write a Python program to create a list comprehension.
45. Write a Python program to create a dictionary comprehension.
46. Write a Python program to create a set comprehension.
47. Write a Python program to create a generator.
48. Write a Python program to use an iterator.
49. Write a Python program to use a decorator.
50. Write a Python program to create a regular expression.
51. Write a Python program to match a pattern using a regular expression.
52. Write a Python program to split a string using a regular expression.
53. Write a Python program to search a string using a regular expression.

54. Write a Python program to create a thread.
55. Write a Python program to create a process.
56. Write a Python program to create a socket.
57. Write a Python program to create a server socket and accept connections.
58. Write a Python program to create a client socket and send data.
59. Write a Python program to create a UDP socket.
60. Write a Python program to create a GUI application using tkinter.
61. Write a Python program to create a button in a GUI application.
62. Write a Python program to create a text box in a GUI application.
64. Write a Python program to create a dialog box in a GUI application.
65. Write a Python program to create a canvas in a GUI application.
66. Write a Python program to create a frame in a GUI application.
67. Write a Python program to create a scrollbar in a GUI application.
68. Write a Python program to create a radiobutton in a GUI application.
69. Write a Python program to create a checkbox in a GUI application.
70. Write a Python program to create a slider in a GUI application.
71. Write a Python program to create a listbox in a GUI application.
72. Write a Python program to create a spinbox in a GUI application.
73. Write a Python program to create a progressbar in a GUI application.
74. Write a Python program to create a treeview in a GUI application.
75. Write a Python program to create a tooltip in a GUI application.
76. Write a Python program to create a date picker in a GUI application.
77. Write a Python program to create a time picker in a GUI application.
78. Write a Python program to create a color picker in a GUI application.
79. Write a Python program to create a file dialog in a GUI application.
80. Write a Python program to create a message box in a GUI application.

81. Write a Python program to create a table in a GUI application.
82. Write a Python program to create a toolbar in a GUI application.
83. Write a Python program to create a status bar in a GUI application.
84. Write a Python program to create a menu bar in a GUI application.
85. Write a Python program to create a notebook in a GUI application.
86. Write a Python program to create a paned window in a GUI application.
87. Write a Python program to create a Toplevel window in a GUI application.
88. Write a Python program to create a splash screen in a GUI application.
89. Write a Python program to create a print dialog in a GUI application.
90. Write a Python program to create a font chooser in a GUI application.
91. Write a Python program to create a layout manager in a GUI application.
92. Write a Python program to create a grid layout in a GUI application.
93. Write a Python program to create a pack layout in a GUI application.
94. Write a Python program to create a place layout in a GUI application.
95. Write a Python program to create a form in a GUI application.
96. Write a Python program to create a login form in a GUI application.
97. Write a Python program to create a registration form in a GUI application.
98. Write a Python program to create a contact form in a GUI application.
99. Write a Python program to create a feedback form in a GUI application.
100. Write a Python program to create a survey form in a GUI application.

100 Prompts by GhatGPT

1. Write a Python program to print "Hello, World!"

```python
 This Python program prints "Hello, World!"
 Print  "Hello, World!" to the console
print("Hello, World!")
```

When you run this program, it will print "Hello, World!" to the console.

2. Write a Python program to add two numbers.

```python
 This Python program adds two numbers
 Get the two numbers  from the user
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
  Add the two numbers
sum = num1 + num2
 Print the sum of the two numbers
print("The sum of", num1, "and", num2, "is", sum)
```

When you run this program, it will ask you to enter two numbers. Once you have entered the two numbers, the program will add them together and print the sum to the console.

For example, if you enter `5` and `10`, the program will print `The sum of 5 and 10 is 15.0`.

3. Write a Python program to calculate the area of a triangle.

~~~~
```python
 This Python program calculates the area of a triangle
 Get  the base and height of the triangle from the user
base = float(input("Enter the base of the triangle: "))
height = float(input(" Enter the height of the triangle: "))
 Calculate the area of the triangle
area = 0.5 * base * height
 Print the area of the triangle
print("The area of the triangle is", area, "square units")
```
~~~~

When you run this program, it will ask you to enter the base and height of the triangle. Once you have entered the base and height, the program will calculate the area of the triangle and print it to the console.

For example, if you enter `5` for the base and `10` for the height, the program will print `The area of the triangle is 25.0 square units`.

4. Write a Python program to swap two variables.

There are a few ways to swap two variables in Python. One way is to use a temporary variable:

~~~~
```python
# This Python program swaps two variables
# Get the two variables from the user
num1 = input("Enter  the first number: ")
num2 = input("Enter the second number: ")
 #Create a temporary variable
temp = num1
# Swap the two variables
num1 = num2
num2 = temp
 Print the swapped variables
print("The first number is now", num1)
~~~~

```
 print("The second number is now", num2)
```

When you run this program, it will ask you to enter two numbers. Once you have entered the two numbers, the program will swap them using a temporary variable and then print the swapped variables to the console.

For example, if you enter `5` and `10`, the program will print `The first number is now 10` and `The second number is now 5`.

Another way to swap two variables in Python is to use tuple unpacking:

```python
# This Python program swaps two variables using tuple unpacking
# Get the  two variables from the user
num1, num2 = input("Enter two numbers, separated by a comma: ").split(",")
 #Swap the two variables
num1, num2 = num2, num1
 #Print the swapped variables
print("The first number is now", num1)
print("The second number is now", num2)
```

When you run this program, it will ask you to enter two numbers, separated by a comma. Once you have entered the two numbers, the program will swap them using tuple unpacking and then print the swapped variables to the console.
For example, if you enter `5,10`, the program will print `The first number is now 10` and `The second number is now 5`.

5. Write a Python program to generate a random number.

```python
# This Python program generates a random number
# Import the random  module
import random
 #Generate a random number between 1 and 100
random_number = random.randint(1, 10 0)
 #Print the random number
print("The random number is", random_number)
```

When you run this program, it will generate a random number between 1 and 100 and print it to the console.
For example, if you run the program, it might print ` The random number is 42`.
You can also use the `random` module to generate random numbers from other distributions, such as the normal distribution or the uniform distribution. For more information, see the `random` module documentation: https://docs.python.org/3/library/random.html

6. Write a Python program to convert kilometers to miles.

```python
# This Python program converts kilometers to miles
# Get the distance  in kilometers from the user
kilometers = float(input("Enter the distance in kilometers: "))
 #Convert kilometers to miles
miles = kilometers *  0.621371
 #Print the distance in miles
print("The distance in miles is", miles)
```

When you run this program, it will ask you to enter the distance in kilometers. Once you have entered the distance in kilometers, the program will convert it to miles and print the distance in miles to the console.
For example, if you enter `100`, the program will print `The distance in miles is 62.1371`.
Note that the conversion factor from kilometers to miles is approximately 0.621371.

7. Write a Python program to check if a number is even or odd.

```python
 #This Python program checks if a number is even or odd
 # Get the number from the user
number = int(input("Enter a number: "))
# Check if the number is even or odd
if number  % 2 == 0:
    print("The number is even")
else:
    print("The number is odd")
```

When you run this program, it will ask you to enter a number. Once you have entered a number, the program will check if the number is even or odd and print the result to the console.
For example, if you enter `10`, the program will print `The number is even`. If you enter `9`, the program will print `The number is odd`. The `%` operator in Python is the modulo operator. It returns the remainder of the division of two numbers. In this program, we are checking if the remainder of the division of the number by 2 is equal to 0. If it is, then the number is even. Otherwise, the number is odd.

8. Write a Python program to check if a year is a leap year or not.

```python
 #This Python program checks if a year is a leap year or  not
 #Get the year from the user
year = int(input("Enter a year: "))
# Check if the year is a leap year
 if year % 4 == 0:
    if year % 100 == 0:
        if year % 400 == 0:
            print("The year", year, "is a leap year")
        else:
            print("The year", year, " is not a leap year")
    else:
        print("The year", year, "is a leap year")
else:
    print("The year", year, "is not a leap year")
```

When you run this program, it will ask you to enter a year. Once you have entered a year, the program will check if the year is a leap year or not and print the result to the console.
For example, if you enter `2000`, the program will print `The year 2000 is a leap year`. If you enter `1900 `, the program will print `The year 1900 is not a leap year`.
A leap year is a year that is divisible by 4 but not divisible by 100, or a year that is divisible by 400. This program checks all of these conditions to determine if a year is a leap year or not.

9. Write a Python program to solve a quadratic equation.

```python
 #This Python program solves a quadratic equation
 #Get the coefficients  of the quadratic equation from the user
a = float(input("Enter the coefficient of x^2: "))
b = float(input("Enter the  coefficient of x: "))
c = float(input("Enter the constant term: "))
# Calculate the discriminant
discriminant = b2 - 4*a*c
 #Check if the discriminant is positive, negative, or zero
if discriminant > 0:
   #  The equation has two  real solutions
    x1 = (-b + discriminant0.5) / (2*a)
    x2 = (-b - discriminant0.5) / (2*a)
    print("The solutions to the equation are", x1, "and", x2)
elif discriminant == 0:
     #The equation has one real solution
    x = -b / (2*a)
    print("The solution to the equation is", x)
else:
     #The equation has no real solutions
    print("The equation has no real solutions")
```

When you run this program, it will ask you to enter the coefficients of the quadratic equation. Once you have entered the coefficients, the program will calculate the discriminant and check if the equation has real solutions or not. If the equation has real solutions, the program will print the solutions to the console. If the equation has no real solutions, the program will print a message to that effect.

For example, if you enter `1`, `2`, and `1` for the coefficients of the quadratic equation, the program will print `The solutions to the equation are -1.0 and -1.0`. If you enter `1`, `-2`, and `1` for the coefficients of the quadratic equation, the program will print `The solution to the equation is 1.0`. If you enter `1`, `2`, and `5` for the coefficients of the quadratic equation, the program will print `The equation has no real solutions`.

10. Write a Python program to calculate the factorial of a number.

```python
 #This Python program calculates the factorial of a number
# Get  the number from the user
number = int(input("Enter a non-negative integer: "))
# Calculate the factorial of the number
factorial =  1
for i in range(1, number + 1):
    factorial *= i
# Print the factorial of the number
print("The factorial of", number, "is", factorial)
```

When you run this program, it will ask you to enter a non-negative integer. Once you have entered a number, the program will calculate the factorial of the number and print it to the console.
For example, if you enter `5`, the program will print `The factorial of 5 is 120`.
The factorial of a non-negative integer `n` is the product of all the positive integers less than or equal to `n`. For example, the factorial of 5 is calculated as follows:

```
```

~~~
5! = 5 * 4 * 3 * 2 * 1 = 120
```
~~~

The factorial of 0 is defined to be 1.

11. Write a Python program to create a list and append items to it.

~~~
```python
# This Python program creates a list and appends items to it
# Create an empty list
my_list = []
# Append items to the list
my_list.append(1)
my_list .append(2)
my_list.append(3)
# Print the list
print("The list is:", my_list)
```
~~~

When you run this program, it will create an empty list, append the numbers 1, 2, and 3 to the list, and then print the list to the console.

The output of the program will be:

~~~
```
The list is: [1, 2, 3]
```
~~~

You can also use the `+` operator to concatenate lists. For example, the following code creates two lists and then concatenates them into a single list:

~~~
```python
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list3 = list1 + list2
print("The concatenated list is:", list3)
```
~~~

The output of this program will be:

~~~
```
The concatenated list is: [1, 2, 3, 4, 5, 6]
```
~~~

12. Write a Python program to sort a list in ascending order.

```python
# This Python program sorts a list in ascending order
# Create  a list of unsorted numbers
unsorted_list = [5, 3, 1, 2, 4]
# Sort the list  in ascending order
sorted_list = sorted(unsorted_list)
 #Print the sorted list
print("The sorted list is:", sorted_list)
```

When you run this program, it will create a list of unsorted numbers, sort the list in ascending order, and then print the sorted list to the console.

The output of the program will be:

```
The sorted list is: [1, 2, 3, 4, 5]
```

You can also use the `sort()` method to sort a list in ascending order. For example, the following code creates a list of unsorted numbers and then sorts the list in ascending order using the `sort()` method:

```python
unsorted_list = [5, 3, 1, 2, 4]
unsorted_list.sort()
print("The sorted list is:", unsorted_ list)
```

The output of this program will be the same as the previous program:

```
The sorted list is: [1, 2, 3, 4, 5]
```

The `sort()` method sorts the list in-place, meaning that it modifies the original list. The `sorted()` function, on the other hand, returns a new list that is sorted, leaving the original list unchanged.

13. Write a Python program to sort a list in descending order.

```python
# This Python program sorts a list in descending order
 #Create  a list of unsorted numbers
unsorted_list = [5, 3, 1, 2, 4]
# Sort the list  in descending order
sorted_list = sorted(unsorted_list, reverse=True)
# Print the sorted list
print("The sorted list is:", sorted_list)
```

When you run this program, it will create a list of unsorted numbers, sort the list in descending order, and then print the sorted list to the console.
The output of the program will be:

```
The sorted list is: [5, 4, 3, 2, 1]
```

You can also use the `sort()` method to sort a list in descending order. To do this, you can pass the `reverse=True` argument to the `sort()` method. For example, the following code creates a list of unsorted numbers and then sorts the list in descending order using the `sort()` method:

```python
unsorted_list = [5, 3, 1,  2, 4]
unsorted_list.sort(reverse=True)
print("The sorted list is:", unsorted_list)
```

The output of this program will be the same as the previous program:

```
The sorted list is: [5, 4, 3, 2, 1]
```

The `sort()` method sorts the list in-place, meaning that it modifies the original list. The `sorted()` function, on the other hand, returns a new list that is sorted, leaving the original list unchanged.

14. Write a Python program to remove duplicates from a list.

```python
 This Python program removes duplicates from a list
 Create a  list with duplicate elements
```

```
list1 = [1, 2, 3, 4, 5, 1, 2, 3]
Create an empty list to store the unique elements
unique_list = []
 Iterate over the list and add each unique element to the unique_list
for x in list1:
   if x not in unique_list:
       unique_list.append(x)
 Print the  unique list
print("The list without duplicates is:", unique_list)
```

When you run this program, it will create a list with duplicate elements, create an empty list to store the unique elements, iterate over the list and add each unique element to the unique_list, and then print the unique list to the console.
The output of the program will be:

```
The list without duplicates is: [1, 2, 3, 4, 5]
```

You can also use the `set()` function to remove duplicates from a list. For example, the following code creates a list with duplicate elements and then uses the `set()` function to remove the duplicates:

```python
list1 = [1, 2, 3, 4, 5, 1, 2, 3]
unique_list = set(list1)
Convert the set back to a list
unique_list = list(unique_list)
print("The list without duplicates is:", unique_list)
```

The output of this program will be the same as the previous program:

```
The list without duplicates is: [1, 2, 3, 4, 5]
```

The `set()` function creates a set from the list, which automatically removes duplicate elements. The `list()` function is then used to convert the set back to a list.

15. Write a Python program to count the number of elements in a list.

```python
 This Python program counts the number of elements in a list
  Create a list of elements
list1 = [1, 2, 3, 4, 5]
 Count the number of elements  in the list
count = len(list1)
Print the count of elements
print("The number of elements in the list is:", count)
```

When you run this program, it will create a list of elements, count the number of elements in the list, and then print the count of elements to the console. The output of the program will be:

```
The number of elements in the list is: 5
```

You can also use the `len()` function to count the number of elements in a list. For example, the following code creates a list of elements and then uses the `len()` function to count the number of elements in the list:

```python
list1 = [1, 2, 3, 4, 5]
count = len(list1)
print("The number of elements in the list is:", count)
```

The output of this program will be the same as the previous program:

```
The number of elements in the list is: 5
```

The `len()` function returns the number of elements in the list.

16. Write a Python program to find the maximum and minimum number in a list.

```python
 This Python program finds the maximum and minimum number in a list
 Create a list of numbers
numbers = [1, 2, 3, 4, 5, -1, -2, - 3]
 Find the maximum and minimum number in the list
max_number = max(numbers)
min_number = min(numbers)
 Print the maximum and minimum number
print("The maximum number in the list is:", max_number)
print("The minimum number in the list is :", min_number)
```

When you run this program, it will create a list of numbers, find the maximum and minimum number in the list, and then print the maximum and minimum number to the console.The output of the program will be:

```
The maximum number in the list is: 5
The minimum number in the list is: -3
```

You can also use the `max()` and `min()` functions to find the maximum and minimum number in a list. For example, the following code creates a list of numbers and then uses the `max()` and `min()` functions to find the maximum and minimum number in the list:

```python
numbers = [1, 2, 3, 4, 5, -1, -2, -3]
max_number = max(numbers)
min_number = min(numbers)
print("The maximum number in the list is:", max_number)
print("The minimum number in the list is:", min_number)
```

The output of this program will be the same as the previous program:

```
The maximum number in the list is: 5
The minimum number in the list is: -3
```

The `max()` and `min()` functions return the maximum and minimum number in the list, respectively.

17. Write a Python program to create a tuple and access its elements.

~~~
```python
 This Python program creates a tuple and accesses its elements
  Create a tuple
my_tuple = (1, 2, 3, 4, 5)
 Access the elements of the tuple
 first_element = my_tuple[0]
second_element = my_tuple[1]
third_element = my_tuple[2]

 Print the elements of the tuple
print("The first element of the tuple is:", first_element)
print("The second element of the tuple  is:", second_element)
print("The third element of the tuple is:", third_element)
```
~~~

When you run this program, it will create a tuple, access the elements of the tuple, and then print the elements of the tuple to the console.
The output of the program will be:

~~~
```

The first element of the tuple is: 1
The second element of the tuple is: 2
The third element of the tuple is: 3
```
~~~

You can also access the elements of a tuple using negative indices. For example, the following code accesses the last element of the tuple using a negative index:

~~~
```python
last_element = my_tuple[-1]
print("The last element of the tuple is:", last_element)
```
~~~

The output of this code will be:

~~~
```

The last element of the tuple is: 5
```
~~~

Tuples are immutable, meaning that you cannot change the elements of a tuple once it has been created.

18. Write a Python program to convert a tuple to a list.

```python
# This Python program converts a tuple to a list
 #Create  a tuple
my_tuple = (1, 2, 3, 4, 5)

 Convert the tuple to a list
my _list = list(my_tuple)
 Print the list
print("The list is:", my_list)
```

When you run this program, it will create a tuple, convert the tuple to a list, and then print the list to the console.
#The output of the program will be :

```

The list is: [1, 2, 3, 4, 5]
```

You can also use the `tuple()` function to convert a list to a tuple. For example, the following code creates a list and then uses the `tuple()` function to convert the list to a tuple:

```python
my_list = [1, 2, 3, 4, 5]
my_tuple = tuple(my_list)
print("The tuple is:", my_tuple)
```

#The output of this program will be:

```
#The  tuple is: (1, 2, 3, 4, 5)
```

Tuples are immutable, meaning that you cannot change the elements of a tuple once it has been created. Lists, on the other hand, are mutable, meaning that you can change the elements of a list after it has been created.

19. Write a Python program to create a dictionary and access its elements.

```python
 This Python program creates a dictionary and accesses its elements

  Create a dictionary
my_dict = {"name": "John Doe", "age": 30, "city": "New York"}
 Access  the elements of the dictionary
name = my_dict["name"]
age = my_dict["age"]
city = my_dict["city"]
 Print the elements of the dictionary
print("Name:", name)
print("Age:", age)
print("City:", city)
```

When you run this program, it will create a dictionary, access the elements of the dictionary, and then print the elements of the dictionary to the console. The output of the program will be:

```
Name: John Doe
Age: 30
City: New York
```

You can also access the elements of a dictionary using the `get()` method. For example, the following code accesses the elements of the dictionary using the `get()` method:

```python
name = my_dict.get("name")
age = my_dict.get("age")
city = my_dict. get("city")

print("Name:", name)
print("Age:", age)
print("City:", city)
```

The output of this program will be the same as the previous program.
The `get()` method is useful for accessing the elements of a dictionary without raising an error if the key does not exist. For example, the following code tries to access the element of the dictionary with the key "country", which does not exist:

```python
country = my_dict.get("country")
print("Country:", country)
```
#This code will not raise an error, and it will print `None` to the console.

Dictionaries are mutable, meaning that you can change the elements of a dictionary after it has been created. For example, the following code changes the value of the "age" key in the dictionary:

```python
my_dict["age"] = 31
print("Age:", my_dict["age"])
```
This code will print `31` to the console.

20. Write a Python program to update the value of a key in a dictionary.

```python
 This Python program updates the value of a key in a dictionary
 Create a dictionary
my_dict = {"name": "John Doe", "age": 30, "city": "New York"}
  Update the value of the "age" key
my_dict["age"] = 31
 Print the updated dictionary
print("Updated dictionary:", my_dict)
```
When you run this program, it will create a dictionary, update the value of the "age" key in the dictionary , and then print the updated dictionary to the console.
The output of the program will be:

```
Updated dictionary: {'name': 'John Doe', 'age': 31, 'city': 'New York'}
```

You can also use the `update()` method to update the value of a key in a dictionary. For example, the following code creates a dictionary and then uses the `update()` method to update the value of the "age" key in the dictionary:

```python
my_dict = {"name": "John Doe", "age": 30, "city": "New  York"}

my_dict.update({"age": 31})

print("Updated dictionary:", my_dict)
```

The output of this program will be the same as the previous program.
The `update()` method can also be used to add new key-value pairs to a dictionary. For example, the following code creates a dictionary and then uses the `update()` method to add a new key-value pair to the dictionary:

```python
my_dict = {"name": "John Doe", "age": 30, "city": "New York"}
my_dict.update({"country": "USA"})
print("Updated dictionary:", my_dict)
```

The output of this program will be:

```
Updated dictionary: {'name': 'John Doe', 'age': 30, 'city': 'New York',
'country': 'USA'}
```

Dictionaries are mutable, meaning that you can change the elements of a dictionary after it has been created.

21. Write a Python program to remove a key from a dictionary.

```python
 #This Python program removes a key from a dictionary
 #Create  a dictionary
my_dict = {"name": "John Doe", "age": 30, "city": "New York"}

# Remove the  "age" key from the dictionary
del my_dict["age"]

 Print the updated dictionary
print("Updated dictionary:", my_dict)
```

When you run this program, it will create a dictionary, remove the "age" key from the dictionary, and then print the updated dictionary to the console. The output of the program will be:

```
Updated dictionary: {'name': 'John Doe', 'city': 'New York'}
```

You can also use the `pop()` method to remove a key from a dictionary. For example, the following code creates a dictionary and then uses the `pop()` method to remove the "age" key from the dictionary:

```python
my_dict = {"name": "John Doe", "age": 30, "city": "New York"}
my_dict.pop("age")
print("Updated dictionary:", my_dict)
```

The output of this program will be the same as the previous program. The `pop()` method also allows you to specify a default value to return if the key does not exist. For example, the following code creates a dictionary and then uses the `pop()` method to remove the "country" key from the dictionary, specifying a default value of "USA":

```python
my_dict = {"name": "John Doe", "age": 30, "city": "New York"}
```

```
country = my_dict.pop("country", "USA")

print("Country:", country)
print("Updated dictionary:", my_dict)
```

The output of this program will be:

```
Country: USA
Updated dictionary: {'name': 'John Doe', 'age': 30, 'city': 'New York'}
```

Dictionaries are mutable, meaning that you can change the elements of a dictionary after it has been created.

22. Write a Python program to sort a dictionary by key.

```python
 #This Python program sorts a dictionary by key

 #Create a  dictionary
my_dict = {"name": "John Doe", "age": 30, "city": "New York"}
# Sort the dictionary  by key
sorted_dict = dict(sorted(my_dict.items()))

# Print the sorted dictionary
print("Sorted dictionary:", sorted_dict)
```

When you run this program, it will create a dictionary, sort the dictionary by key, and then print the sorted dictionary to the console.
The output of the program will be:

```
Sorted dictionary: {'age': 30, 'city': 'New York', 'name': 'John Doe'}
```

~~~
```
~~~

You can also use the `sorted()` function to sort a dictionary by key. For example, the following code creates a dictionary and then uses the `sorted()` function to sort the dictionary by key:

```python
my_dict = {"name": "John Doe", "age": 30, "city": "New York"}

sorted_dict = sorted(my_dict.items())

print("Sorted dictionary:", sorted _dict)
```

The output of this program will be the same as the previous program.

The `sorted()` function returns a list of tuples, where each tuple contains a key-value pair from the dictionary. You can then use the `dict()` function to convert the list of tuples back into a dictionary.

Dictionaries are mutable, meaning that you can change the elements of a dictionary after it has been created. However, the `sorted()` function and the `dict()` function both return new objects, so the original dictionary will not be sorted.

23. Write a Python program to sort a dictionary by value.

```python
 #This Python program sorts a dictionary by value
# Create a  dictionary
my_dict = {"name": "John Doe", "age": 30, "city": "New York"}

# Sort the dictionary  by value
sorted_dict = dict(sorted(my_dict.items(), key=lambda item: item[1]))
```

```
#Print the sorted dictionary
print("Sorted dictionary:", sorted_dict)
```

When you run this program, it will create a dictionary, sort the dictionary by value, and then print the sorted dictionary to the console.

The output of the program will be:

```
Sorted dictionary: {'age': 30, 'name': 'John Doe', 'city': 'New York'}
```

The `sorted()` function is used to sort the dictionary by value. The `key` argument is used to specify the function that should be used to sort the items. In this case, the `key` argument is a lambda function that returns the value of the item.

The `dict()` function is then used to convert the list of tuples back into a dictionary.

Dictionaries are mutable , meaning that you can change the elements of a dictionary after it has been created. However, the `sorted()` function and the `dict()` function both return new objects, so the original dictionary will not be sorted.

If you want to sort the dictionary in descending order, you can use the `reverse=True` argument to the `sorted()` function. For example, the following code creates a dictionary and then uses the `sorted()` function to sort the dictionary by value in descending order:

```python
my_dict = {"name": "John Doe", "age": 30, "city": "New York"}
sorted_dict = dict(sorted(my_dict.items(), key=lambda item: item[1],
reverse=True))
print("Sorted dictionary:", sorted_dict)
```

#The output of this program will be:

```
Sorted dictionary: {'city': 'New York', 'name': 'John Doe', 'age': 30}
```

24. Write a Python program to create a set and add items to it.

```python
 #This Python program creates a set and adds items to it
  #Create a set
my_set = set()

 Add items to the set
my_set.add(1)
my_set. add(2)
my_set.add(3)

 #Print the set
print("The set is:", my_set)
```

When you run this program, it will create a set, add the numbers 1, 2, and 3 to the set, and then print the set to the console.

The output of the program will be:

```
The set is: {1, 2, 3}
```

Sets are unordered collections of unique elements. This means that the order of the elements in the set is not guaranteed, and that each element in the set can only appear once.

You can also use the `set()` function to create a set from a list. For example, the following code creates a list and then uses the `set()` function to create a set from the list:

~~~
```python
my_list = [1, 2, 3 , 1, 2, 3]

my_set = set(my_list)

print("The set is:", my_set)
```
~~~

The output of this program will be:

~~~
```
The set is: {1, 2, 3}
```
~~~

The `set()` function will automatically remove duplicate elements from the list when it creates the set.

Sets are mutable, meaning that you can change the elements of a set after it has been created. For example, the following code creates a set and then adds the number 4 to the set:

~~~
```python
my_set = {1, 2, 3}
my_set.add(4)
print("The set is:", my_set)
```
~~~

The output of this program will be:

~~~
```
The set is: {1, 2, 3, 4}
```
~~~

25. Write a Python program to remove an item from a set.

```python
 #This Python program removes an item from a set
 #Create  a set
my_set = {1, 2, 3, 4, 5}

 Remove the number 3 from the set
my_set.remove(3)

# Print the set
print("The set is:", my_set)
```

When you run this program, it will create a set, remove the number 3 from the set, and then print the set to the console.

The output of the program will be:

```
The set is: {1, 2, 4, 5}
```

You can also use the `discard()` method to remove an item from a set. The `discard()` method is similar to the `remove()` method, but it does not raise an error if the item is not in the set.

For example, the following code creates a set and then uses the `discard()` method to remove the number 3 from the set:

```python
my_set = {1, 2, 3, 4, 5}

my_set. discard(3)

print("The set is:", my_set)
```

The output of this program will be the same as the previous program.

Sets are mutable, meaning that you can change the elements of a set after it has been created.

If you try to remove an item from a set that is not in the set, the `remove()` method will raise an error, while the `discard()` method will do nothing.

26. Write a Python program to perform set operations like union, intersection, difference, and symmetric difference.

```python
 This Python program performs set operations like union, intersection,
difference , and symmetric difference

 Create two sets
set1 = {1, 2, 3, 4, 5}
set2 =  {3, 4, 5, 6, 7}

 Find the union of the two sets
union_set = set1.union(set2)

 Find the intersection of the two sets
intersection_set = set1.intersection(set2)

 Find the difference  of the two sets
difference_set = set1.difference(set2)

 Find the symmetric difference of the two sets
symmetric_difference_set = set1.symmetric_difference(set2)

 Print the results
print("Union:", union_set)
print("Intersection:", intersection_set)
print("Difference:", difference_set)
print("Symmetric difference:", symmetric_difference_set)
```

When you run this program, it will create two sets, find the union, intersection, difference, and symmetric difference of the two sets, and then print the results to the console.

The output of the program will be:

```
Union: {1, 2, 3, 4, 5, 6, 7}
Intersection: {3, 4, 5}
Difference: {1, 2}
Symmetric difference: {1, 2, 6, 7}
```

Union: The union of two sets is a new set that contains all the elements from both sets.

Intersection: The intersection of two sets is a new set that contains only the elements that are common to both sets.

Difference: The difference of two sets is a new set that contains the elements that are in the first set but not in the second set.

Symmetric difference: The symmetric difference of two sets is a new set that contains the elements that are in one set but not in the other set.

Sets are mutable, meaning that you can change the elements of a set after it has been created. However, the set operations `union()`, `intersection()`, `difference()`, and `symmetric_difference()` all return new sets, so the original sets will not be modified.

27. Write a Python program to create a function and call it.

```python
 This Python program creates a function and calls it

 Define  a function
def my_function(x):
   return x * 2

 Call the function
result = my_function(5)

  Print the result
print("The result is:", result)
```

When you run this program, it will define a function called `my_function()`, call the function with the argument `5`, and then print the result to the console.

The output of the program will be:

```
 The result is: 10
```

Functions are blocks of code that can be reused. They can be defined using the `def` keyword, followed by the function name and the function arguments. The function body is indented.

To call a function, you simply use the function name followed by the function arguments.

In this example, the `my_function()` function takes one argument, `x`, and returns the value of `x` multiplied by 2. The function is called with the argument `5`, and the result is printed to the console.

You can define functions anywhere in your Python program, but it is generally considered good practice to define all of your functions at the beginning of the program.

28. Write a Python program to create a function with arguments and return a value.

```python
 This Python program creates a function with arguments and returns a value

 Define a function
def my_function(x, y):
    return x + y

 Call the function
result = my_function (5, 10)

 Print the result
print("The result is:", result)
```

When you run this program, it will define a function called `my_function()`, call the function with the arguments `5` and `10`, and then print the result to the console .

The output of the program will be:

```
The result is: 15
```

Functions are blocks of code that can be reused. They can be defined using the `def` keyword, followed by the function name and the function arguments. The function body is indented.

To call a function, you simply use the function name followed by the function arguments.

In this example, the `my_function()` function takes two arguments, `x` and `y`, and returns the value of `x` plus `y`. The function is called with the arguments `5` and `10`, and the result is printed to the console.

You can define functions anywhere in your Python program, but it is generally considered good practice to define all of your functions at the

beginning of the program.

Functions can also return multiple values. To do this, you can use a tuple. For example, the following function returns the sum and product of two numbers:

```python
def my_function(x, y):
   return x + y, x * y

 Call the function
result = my_function(5, 10)

 Unpack the tuple
sum, product = result

 Print the results
print("The sum is:", sum)
print("The product is:", product)
```

The output of this program will be:

```

The sum is: 15
The product is: 50
```

29. Write a Python program to create a lambda function.

```python
 This Python program creates a lambda function

 Define a lambda  function
my_function = lambda x: x * 2

 Call the lambda function
result = my_function(5)
```

 Print the result
print("The result is:", result)
```

When you run this program, it will define a lambda function called `my_function()`, call the lambda function with the argument `5`, and then print the result to the console.

The output of the program will be:

```
The  result is: 10

Lambda functions are anonymous functions, meaning that they do not have a name. They are defined using the `lambda` keyword, followed by the function arguments and a colon. The function body is indented.

To call a lambda function, you simply use the lambda function expression.

In this example, the `my_function()` lambda function takes one argument, `x`, and returns the value of `x` multiplied by 2. The lambda function is called with the argument `5`, and the result is printed to the console.

Lambda functions are often used for simple tasks that do  not require a full-fledged function definition. For example, you could use a lambda function to sort a list of numbers:

```python
numbers = [1, 5, 3, 2, 4]

sorted_numbers = sorted(numbers, key=lambda x: x)

print("Sorted numbers:", sorted_numbers)
```

The output of this program will be:

```
Sorted numbers: [1, 2, 3, 4, 5]
```

In this example, the `lambda x: x` lambda function is used to sort the list of numbers in ascending order. The `sorted()` function takes two arguments: the list to be sorted and a key function. The key function is used to specify how the elements of the list should be compared. In this case, the key function is the lambda function `lambda x: x`, which simply returns the value of `x`. This means that the elements of the list will be sorted in ascending order based on their values.

30. Write a Python program to create a class and an object.

```python
 This Python program creates a class and an object

 Define  a class
class MyClass:
    def __init__(self, name):
        self.name = name

    def greet(self):
        print("Hello, my name is", self.name)

 Create an object of the class
my_object = MyClass("John Doe")

 Call the greet() method of the object
my_object.greet()
```

When you run this program, it will define a class called `My Class`, create an object of the class, and then call the `greet()` method of the object.

The output of the program will be:

```
Hello, my name is John Doe
```

Classes are used to create objects. An object is an instance of a class.

To define a class, you use the `class` keyword, followed by the class name and a colon. The class body is indented.

The `__init__()` method is a special method that is called when an object is created. It is used to initialize the object's attributes.

In this example, the `My Class` class has one attribute, `name`. The `__init__()` method takes one argument, `name`, and assigns it to the `name` attribute of the object.

The `greet()` method is a regular method that can be called on objects of the class. It simply prints a message to the console.

To create an object of a class, you use the `Classname()` syntax. For example, the following code creates an object of the `MyClass` class:

```python
my_object = MyClass("John Doe")
```

Once you have created an object, you can call its methods using the dot operator. For example, the following code calls the `greet()` method of the `my_object` object:

```python
my_object.greet()
```

This will print the following message to the console:

```
Hello, my name is John Doe
```

31. Write a Python program to create a class with a method and call it.

```python
 This Python program creates a class with a method and calls it

 Define a class
class MyClass:
    def __init__(self, name):
        self.name = name

    def greet(self ):
        print("Hello, my name is", self.name)

 Create an object of the class
my_object = MyClass("John Doe")

 Call the greet() method of the object
my_object.greet()
```

When you run this program, it will define a class called `MyClass`, create an object of the class, and then call the `greet()` method of the object.

The output of the program will be:

```
Hello, my name is John Doe
```

Classes are used to create objects. An object is an instance of a class.

To define a class, you use the `class` keyword, followed by the class name and

a colon. The class body is indented.

The `__init__()` method is a special method that is called when an object is created. It is used to initialize the object's attributes.

In this example, the `MyClass` class has one attribute, `name`. The `__init__()` method takes one argument, `name`, and assigns it to the `name` attribute of the object.

The `greet()` method is a regular method that can be called on objects of the class. It simply prints a message to the console.

To create an object of a class, you use the `Classname()` syntax. For example, the following code creates an object of the `MyClass` class:

```python
my_object = MyClass("John Doe")
```

Once you have created an object, you can call its methods using the dot operator. For example, the following code calls the `greet()` method of the `my_object` object:

```python
my_object.greet()
```

This will print the following message to the console:

```

Hello, my name is John Doe
```

In this example, the `greet()` method is called on the `my_object` object. The `self` parameter is a reference to the object itself. This allows the method to access the object's attributes and methods.

32. Write a Python program to create a class with a constructor.

```python
 This Python program creates a class with a constructor

 Define  a class
class MyClass:
    def __init__(self, name):
        self.name = name

    def greet(self):
         print("Hello, my name is", self.name)

 Create an object of the class
my_object = MyClass("John Doe")

 Call the greet() method of the object
my_object.greet()
```

When you run this program, it will define a class called `My Class`, create an object of the class, and then call the `greet()` method of the object.

The output of the program will be:

```
Hello, my name is John Doe
```

Classes are used to create objects. An object is an instance of a class.

To define a class, you use the `class` keyword, followed by the class name and a colon. The class body is indented.

The `__init__()` method is a special method that is called when an object is created. It is used to initialize the object's attributes.

In this example, the `My Class` class has one attribute, `name`. The `__init__()` method takes one argument, `name`, and assigns it to the `name` attribute of the object.

The `greet()` method is a regular method that can be called on objects of the class. It simply prints a message to the console.

To create an object of a class, you use the `Classname()` syntax. For example, the following code creates an object of the `MyClass` class:

```python
my_object = MyClass("John Doe")
```

Once you have created an object, you can call its methods using the dot operator. For example, the following code calls the `greet()` method of the `my_object` object:

```python
my_object.greet()
```

This will print the following message to the console:

```

Hello, my name is John Doe
```

In this example, the `__init__()` method is the constructor for the `MyClass` class. The constructor is called when an object of the class is created. The constructor takes one argument, `name`, and assigns it to the `name` attribute of the object.

33. Write a Python program to create a class with inheritance.

```python
 This Python program creates a class with inheritance

 Define a  parent class
class Parent:
    def __init__(self, name):
        self.name = name

    def greet(self):
         print("Hello, my name is", self.name)

 Define a child class
class Child(Parent):
    def __init__(self, name, age):
        super().__init__(name)
        self.age = age

    def greet(self):
        print("Hello,  my name is", self.name, "and I am", self.age, "years old")

 Create an object of the child class
my_object = Child("John Doe", 30)

 Call the greet() method of the object
my_object.greet()
```

When you run this program, it will define a parent class called `Parent` and a child class called `Child`. It will then create an object of the `Child` class and call the `greet()` method of the object.

The output of the program will be:

```
Hello, my name is  John Doe and I am 30 years old
```

Inheritance is a way of creating new classes from existing classes. The new class, called the child class, inherits the attributes and methods of the existing class, called the parent class.

In this example, the `Child` class inherits from the `Parent` class. This means that the `Child` class has all of the attributes and methods of the `Parent` class, plus any additional attributes and methods that are defined in the `Child` class.

In this example, the `Child` class has one additional attribute, `age`, and one additional method, `greet()`. The `greet()` method of the `Child` class overrides the `greet()` method of the `Parent` class. This means that when the `greet()` method is called on an object of the `Child` class, the `greet()` method of the `Child` class will be called instead of the `greet()` method of the `Parent` class.

To create an object of a child class, you use the `Classname()` syntax. For example, the following code creates an object of the `Child` class:

```python
my_object = Child("John Doe", 30 )
```

Once you have created an object of a child class, you can call its methods using the dot operator. For example, the following code calls the `greet()` method of the `my_object` object:

```python
my_object.greet()
```

This will print the following message to the console:

```
Hello, my name is John Doe and I am 30 years old
```

34. Write a Python program to handle an exception using try-except block.

~~~
```python
 This Python program handles an exception using a try-except block

 Define a function that may raise an exception
def divide(x, y):
   return x / y

 Try to call the divide()  function with valid arguments
try:
   result = divide(10, 2)
   print("The result is:", result)
except ZeroDivisionError:
   print("Error: division by zero")

 Try to call the divide() function with invalid arguments
try:
   result =  divide(10, 0)
   print("The result is:", result)
except ZeroDivisionError:
   print("Error: division by zero")
```
~~~

When you run this program, it will first call the `divide()` function with valid arguments. The `divide()` function will return the result of dividing 10 by 2, which is 5. The program will then print the result to the console.

Next, the program will try to call the `divide()` function with invalid arguments. The `divide()` function will raise a `ZeroDivisionError` exception because you cannot divide by zero . The program will catch the exception using the `except` block and print an error message to the console.

The output of the program will be:

```
The result is: 5.0
Error: division by zero
```

The `try-except` block is used to handle exceptions in Python. The `try` block contains the code that may raise an exception. The `except` block contains the code that will be executed if an exception is raised in the `try` block.

In this example, the `try` block contains the code that calls the `divide()` function with valid and invalid arguments. The `except` block contains the code that prints an error message if a `ZeroDivisionError` exception is raised.

You can also use the `else` block to handle code that should be executed if no exception is raised in the `try` block. For example, the following code uses the `else` block to print a message if the `divide()` function is called with valid arguments:

```python
def divide(x, y):
   return x / y

try:
   result = divide(10, 2)
   print("The result is:", result)
except Zero DivisionError:
   print("Error: division by zero")
else:
   print("No exception was raised")
```

The output of this program will be:

```
The result is: 5.0
No exception was raised
```

35. Write a Python program to handle multiple exceptions.

```python
 This Python program handles multiple exceptions

 Define a function that  may raise multiple exceptions
def divide(x, y):
    if y == 0:
        raise ZeroDivisionError("division by zero")
     elif x < 0 or y < 0:
        raise ValueError("negative numbers not allowed")
    else:
        return x / y

 Try to call the divide() function with valid and invalid arguments
try:
    result = divide(10, 2)
    print ("The result is:", result)

    result = divide(-1, 2)
    print("The result is:", result)

    result = divide(10, 0)
    print("The result is:", result)
except ZeroDivisionError:
    print("Error: division by zero")
except ValueError:
    print("Error: negative numbers not allowed")
```

When you run this program, it will first call the `divide()` function with valid arguments. The `divide()` function will return the result of dividing 10 by 2, which is 5. The program will then print the result to the console.

Next, the program will try to call the `divide()` function with invalid arguments. The `divide()` function will raise a `ValueError` exception because negative numbers are not allowed. The program will catch the exception using the `except` block and print an error message to the console.

Finally, the program will try to call the `divide()` function with invalid arguments. The `divide()` function will raise a `ZeroDivisionError` exception because you cannot divide by zero. The program will catch the exception using the `except` block and print an error message to the console.

The output of the program will be:

```
The result is: 5.0
Error: negative numbers not allowed
Error: division by zero
```

The `try-except` block is used to handle exceptions in Python. The `try` block contains the code that may raise an exception. The `except` block contains the code that will be executed if an exception is raised in the `try` block.

In this example, the `try` block contains the code that calls the `divide()` function with valid and invalid arguments. The `except` block contains the code that prints an error message if a `ZeroDivisionError` or `ValueError` exception is raised.

You can use multiple `except` blocks to handle different types of exceptions. The `except` blocks are checked in order, and the first `except` block that matches the exception that is raised will be executed.

36. Write a Python program to raise an exception.

```python
 This Python program raises an exception

 Define a function that  raises an exception
def divide(x, y):
   if y == 0:
       raise ZeroDivisionError("division by zero")
```

```
    else:
       return x / y

 Try to call the divide() function with valid and invalid arguments
try:
   result = divide(10, 2)
   print("The result is:", result)

   result = divide(10, 0)
   print("The  result is:", result)
except ZeroDivisionError:
   print("Error: division by zero")
```

When you run this program, it will first call the `divide()` function with valid arguments. The `divide()` function will return the result of dividing 10 by 2, which is 5. The program will then print the result to the console.

Next, the program will try to call the `divide()` function with invalid arguments. The `divide()` function will raise a `ZeroDivisionError` exception because you cannot divide by zero. The program will catch the exception using the `except` block and print an error message to the console.

The output of the program will be:

```
The result is: 5.0
Error: division by zero
```

The `raise` keyword is used to raise an exception in Python. The `raise` keyword is followed by the exception class and a message.

In this example, the `divide()` function raises a `ZeroDivisionError` exception if the second argument is zero. The message for the exception is "division by zero".

You can raise any type of exception in Python. The built-in exception classes are defined in the `exceptions` module. You can also create your own custom exception classes.

Raising an exception is a way to handle errors in your program. When an exception is raised, the program will stop executing the current code and jump to the nearest `except` block that matches the exception that was raised.

37. Write a Python program to create a file and write to it.

```python
 This Python program creates a file and writes to it

  Open a file for writing
file = open("myfile.txt", "w")

 Write some data to the file
file.write("Hello,  world!")

 Close the file
file.close()
```

When you run this program, it will create a new file called `myfile.txt` and write the string "Hello, world!" to the file.

The `open()` function is used to open a file. The first argument to the `open()` function is the name of the file to open. The second argument is the mode to open the file in. In this example, we are opening the file in write mode, which means that we can write data to the file.

The `write()` method is used to write data to a file. The argument to the `write()` method is the data to write to the file. In this example, we are writing the string "Hello, world!" to the file.

The `close()` method is used to close a file. It is important to close a file after you are finished writing to it.

You can also use the `with` statement to open a file. The `with` statement automatically closes the file when you are finished with it. For example, the following code uses the `with` statement to open a file and write to it:

```python
with open("myfile.txt", "w") as file:
   file.write("Hello, world!")
```

The `with` statement is a more concise way to open and close a file. It is also safer because it ensures that the file is always closed, even if an exception is raised.

38. Write a Python program to read a file.

```python
 This Python program reads a file

 Open a file for  reading
file = open("myfile.txt", "r")

 Read the contents of the file
data = file.read()

 Print the  contents of the file
print(data)

 Close the file
file.close()
```

When you run this program, it will open the file `myfile.txt` for reading, read the contents of the file, and then print the contents of the file to the console.

The `open ()` function is used to open a file. The first argument to the `open()` function is the name of the file to open. The second argument is the mode to

open the file in. In this example, we are opening the file in read mode, which means that we can read data from the file.

The `read()` method is used to read data from a file. The `read()` method returns a string containing the contents of the file.

The `print()` function is used to print data to the console.

The `close()` method is used to close a file. It is important to close a file after you are finished reading from it.

You can also use the `with` statement to open a file. The `with` statement automatically closes the file when you are finished with it. For example, the following code uses the `with` statement to open a file and read from it:

```python
with open("myfile.txt", "r") as file:
   data = file.read()
   print(data)
```

The `with` statement is a more concise way to open and close a file. It is also safer because it ensures that the file is always closed, even if an exception is raised.

39. Write a Python program to append text to a file.

```python
 This Python program appends text to a file

 Open  a file for appending
file = open("myfile.txt", "a")

 Write some data to the file
file.write("\nThis  is some new text.")

 Close the file
```

file.close()
**```**

When you run this program, it will open the file `myfile.txt` for appending, write the string "\nThis is some new text." to the file, and then close the file.

The `open()` function is used to open a file. The first argument to the `open()` function is the name of the file to open. The second argument is the mode to open the file in. In this example, we are opening the file in append mode, which means that we can add data to the end of the file.

The `write()` method is used to write data to a file. The argument to the `write()` method is the data to write to the file. In this example, we are writing the string "\nThis is some new text." to the file.

The `close()` method is used to close a file. It is important to close a file after you are finished writing to it.

You can also use the `with` statement to open a file. The `with` statement automatically closes the file when you are finished with it. For example, the following code uses the `with` statement to open a file and append to it:

```python
with open("myfile.txt", "a") as file:
    file.write("\nThis is some new text.")
```

The `with` statement is a more concise way to open and close a file. It is also safer because it ensures that the file is always closed, even if an exception is raised.

40. Write a Python program to delete a file.

```python
 This Python program deletes a file

 Import the os module
```

```python
import os

 Delete the file
os.remove("myfile.txt")
```

When you run this program, it will delete the file ` myfile.txt`.

The `os.remove()` function is used to delete a file. The argument to the `os.remove()` function is the name of the file to delete.

You can also use the `os.unlink()` function to delete a file. The `os.unlink()` function is similar to the `os.remove()` function, but it can also be used to delete symbolic links.

For example, the following code uses the `os.unlink()` function to delete a file:

```python
import os
os.unlink("myfile.txt")
```

Both the `os.remove()` function and the `os.unlink()` function will raise an error if the file does not exist.
You can use the `os.path.exists()` function to check if a file exists before you try to delete it. For example, the following code uses the `os.path.exists()` function to check if the file `myfile.txt` exists before it tries to delete it:

```python
import os

if os.path.exists("myfile.txt"):
   os.remove("myfile.txt")
else:
   print("The file does not exist.")
```

This code will print the message "The file does not exist." if the file `myfile.txt` does not exist. Otherwise, it will delete the file.

41. Write a Python program to create a module and import it.

To create a module in Python, you simply need to create a Python file with a `.py` extension. For example, you could create a module called `mymodule.py` with the following code:

```python
def greet (name):
   print(f"Hello, {name}!")
```

To import this module into another Python program, you can use the `import` statement. For example, the following code imports the `mymodule` module:

```python
import mymodule

mymodule.greet("John  Doe")
```

This code will print the following message to the console:

```
Hello, John Doe!
```

You can also import specific functions from a module using the `from` statement. For example, the following code imports the `greet()` function from the `mymodule` module:

```python
from mymodule import greet

greet("John Doe")
```

This code will also print the following message to the console:

```
Hello, John Doe!
```

Modules are a powerful way to organize and reuse code in Python. You can create your own modules or import modules that have been created by other people.

Here are some of the benefits of using modules:

*** Code reusability: Modules allow you to reuse code across multiple programs. This can save you time and effort, and it can also help to ensure that your code is consistent.**
*** Organization: Modules can help you to organize your code into logical units. This can make it easier to find and maintain your code.**
*** Encapsulation: Modules can help you to encapsulate your code, which can make it more secure and easier to maintain.**

Modules are an essential part of Python programming. By using modules, you can create more powerful and maintainable programs.

42. Write a Python program to install a package using pip.

To install a package using pip, you can use the following command:

```
pip install package_name
```

For example, to install the `requests` package, you would use the following command:

```
pip install  requests
```

You can also install packages from a specific URL or from a local directory. For example, to install the `requests` package from a URL, you would use the following command:

```
pip install https://pypi.org/project/requests/
```

To install the `requests ` package from a local directory, you would use the following command:

```
pip install /path/to/requests
```

Once you have installed a package, you can import it into your Python programs using the `import` statement. For example, to import the `requests` package, you would use the following code:

```python
import requests
```

You can then use the `requests` package to make HTTP requests. For example, the following code uses the `requests` package to make a GET request to the Google homepage:

```python
import requests

response = requests.get("https:// www.google.com")

print(response.text)
```

This code will print the HTML of the Google homepage to the console.

Pip is a powerful tool that allows you to easily install and manage Python packages. By using pip, you can quickly and easily add new features to your Python programs.

Here are some of the benefits of using pip:

*** Easy installation: Pip makes it easy to install Python packages. You can install packages with just a few simple commands.**
*** Package management: Pip can help you to manage your Python packages. You can use pip to install, uninstall, and update packages.**
*** Package discovery: Pip can help you to discover new Python packages. You can use pip to search for packages and read their documentation.**

Pip is an essential tool for Python developers. By using pip, you can easily install and manage Python packages, and you can quickly and easily add new features to your Python programs.

43. Write a Python program to create a virtual environment.

To create a virtual environment in Python, you can use the following steps:

1. Install the `venv` module. You can do this using the following command:

```
pip install venv
```

2. Create a directory for your virtual environment. For example, you could create a directory called `my_venv`.

```
mkdir my_venv
```

3. Activate the virtual environment. You can do this using the following command:

```
venv/bin/activate
```

This will activate the virtual environment and change your shell prompt to indicate that you are in the virtual environment.

4. Install the packages that you need for your project. You can do this using the `pip` command. For example, to install the `requests` package, you would use the following command:

```
pip install requests
```

5. Deactivate the virtual environment when you are finished working on your project. You can do this using the following command:

```
deactivate
```

This will deactivate the virtual environment and change your shell prompt back to normal.

Virtual environments are a useful way to isolate your Python projects from each other. This can help to prevent conflicts between different versions of packages and it can also make it easier to manage your Python projects.

Here are some of the benefits of using virtual environments:

*** Isolation: Virtual environments allow you to isolate your Python projects from each other. This can help to prevent conflicts between different versions of packages and it can also make it easier to manage your Python projects.**
*** Dependency management: Virtual environments allow you to manage the dependencies for your Python projects. You can install different versions of**

packages in different virtual environments, which can be useful for testing and development.
*** Portability: Virtual environments are portable. You can easily move a virtual environment from one computer to another.**

Virtual environments are an essential tool for Python developers. By using virtual environments, you can easily isolate your Python projects from each other, manage your Python dependencies, and easily move your Python projects from one computer to another.

44. Write a Python program to create a list comprehension.

A list comprehension is a concise way to create a list in Python. It consists of a series of expressions enclosed in square brackets. For example, the following code creates a list of the numbers from 1 to 10 using a list comprehension:

```python
numbers = [x for x in range(1, 11)]
```

This code is equivalent to the following code:

```python
numbers = []
for x in range(1, 11):
   numbers.append(x)
```

List comprehensions can be used to create lists of all sorts of things. For example, the following code creates a list of the squares of the numbers from 1 to 10:

```python
squares = [x * x for x in range(1, 11)]
```

This code is equivalent to the following code:

```python
squares = []
for x in range(1, 11):
   squares.append(x * x)
```

List comprehensions can also be used to filter lists. For example, the following code creates a list of the even numbers from 1 to 10:

```python
even_numbers = [x for x in range(1, 11) if x % 2 == 0]
```

This code is equivalent to the following code:

```python
even_numbers = []
for x in range(1, 11):
   if x % 2 == 0:
       even_numbers.append(x)
```

List comprehensions are a powerful tool that can be used to create and manipulate lists in Python. They are a concise and readable way to write code.

Here are some of the benefits of using list comprehensions:

*** Conciseness: List comprehensions are a concise way to write code. They can often be used to replace multiple lines of code with a single line of code.**
*** Readability: List comprehensions are easy to read and understand. They make it clear what the code is doing.**
*** Versatility: List comprehensions can be used to create and manipulate lists of all sorts of things. They can be used to filter lists, create new lists from existing lists, and perform other operations on lists.**

List comprehensions are an essential tool for Python developers. By using list comprehensions, you can write more concise, readable, and versatile code.

45. Write a Python program to create a dictionary comprehension.

A dictionary comprehension is a concise way to create a dictionary in Python. It consists of a series of key-value pairs enclosed in curly braces. For example, the following code creates a dictionary of the numbers from 1 to 10 and their squares using a dictionary comprehension:

```python
numbers = {x: x * x for x in range(1, 11)}
```

This code is equivalent to the following code:

```python
numbers = {}
for x in range(1, 11):
    numbers[x] = x * x
```

Dictionary comprehensions can be used to create dictionaries of all sorts of things. For example, the following code creates a dictionary of the names of the months and their corresponding numbers:

```python
months = {
   "January": 1,
   "February": 2,
   "March": 3,
   "April": 4,
   "May": 5,
   "June": 6,
   "July": 7,
   "August": 8,
   "September": 9,
   " October": 10,
   "November": 11,
   "December": 12
}
```

This code is equivalent to the following code:

```python
months = {}
months["January"] = 1
months["February"] = 2
months["March"] = 3
months["April"] = 4
months["May"] = 5
months["June"] = 6
months["July"] = 7
months["August"] = 8
months["September"] = 9
months["October"] = 10
months["November"] = 11
months["December"] = 12
```

Dictionary comprehensions are a powerful tool that can be used to create and manipulate dictionaries in Python. They are a concise and readable way to write code.

Here are some of the benefits of using dictionary comprehensions:

*** Conciseness: Dictionary comprehensions are a concise way to write code. They can often be used to replace multiple lines of code with a single line of code.**
*** Readability: Dictionary comprehensions are easy to read and understand. They make it clear what the code is doing.**
*** Versatility : Dictionary comprehensions can be used to create and manipulate dictionaries of all sorts of things. They can be used to filter**

dictionaries, create new dictionaries from existing dictionaries, and perform other operations on dictionaries.

Dictionary comprehensions are an essential tool for Python developers. By using dictionary comprehensions, you can write more concise, readable, and versatile code.

46. Write a Python program to create a set comprehension.

A set comprehension is a concise way to create a set in Python. It consists of a series of expressions enclosed in curly braces. For example, the following code creates a set of the numbers from 1 to 10 using a set comprehension:

```python
numbers = {x for x in range(1, 11)}
```

This code is equivalent to the following code:

```python
numbers = set()
for x in range(1, 11):
   numbers.add(x)
```

Set comprehensions can be used to create sets of all sorts of things. For example, the following code creates a set of the vowels in the English alphabet:

```python
vowels = {"a", "e", "i", "o", "u"}
```

This code is equivalent to the following code:

```python
vowels = set()
```

```
vowels.add("a")
vowels.add("e")
vowels.add("i")
vowels.add("o")
vowels.add("u")
```

Set comprehensions can also be used to filter sets . For example, the following code creates a set of the even numbers from 1 to 10:

```python
even_numbers = {x for x in range(1, 11) if x % 2 == 0}
```

This code is equivalent to the following code:

```python
even_numbers = set()
for x in range(1, 11):
   if x % 2 == 0:
       even_numbers.add(x)
```

Set comprehensions are a powerful tool that can be used to create and manipulate sets in Python. They are a concise and readable way to write code.

Here are some of the benefits of using set comprehensions:

*** Conciseness: Set comprehensions are a concise way to write code. They can often be used to replace multiple lines of code with a single line of code.**
*** Readability: Set comprehensions are easy to read and understand. They make it clear what the code is doing.**
*** Versatility: Set comprehensions can be used to create and manipulate sets of all sorts of things. They can be used to filter sets, create new sets from existing sets, and perform other operations on sets.**

Set comprehensions are an essential tool for Python developers. By using set comprehensions, you can write more concise, readable, and versatile code.

47. Write a Python program to create a generator.

A generator is a function that returns a sequence of values, one at a time . Generators are created using the `yield` keyword. For example, the following code creates a generator that returns the numbers from 1 to 10:

```python
def my_generator():
    for x in range(1, 11):
        yield x
```

To use a generator, you can call it like a regular function. However, instead of returning a value, the generator will yield a sequence of values. You can iterate over the values in the sequence using a `for` loop. For example, the following code uses a `for` loop to iterate over the values in the `my_generator()` generator:

```python
for x in my_generator():
    print(x)
```

This code will print the numbers from 1 to 10.

Generators are useful for creating sequences of values that are too large to store in memory all at once. For example, the following code creates a generator that returns the Fibonacci sequence:

```python
def fibonacci():
    a, b = 0, 1
     while True:
        yield a
    a, b = b, a + b
```

This generator can be used to generate the Fibonacci sequence as large as you want. For example, the following code uses a `for` loop to generate the first 100 numbers in the Fibonacci sequence:

```python
for x in fibonacci():
    print(x)
```

This code will print the first 100 numbers in the Fibonacci sequence.

Generators are a powerful tool that can be used to create sequences of values in a memory-efficient way. They are also easy to use and can be used in a variety of ways.

Here are some of the benefits of using generators:

* Memory efficiency: Generators are memory efficient because they only store the current value in the sequence. This means that they can be used to generate sequences of values that are too large to store in memory all at once.
* Ease of use: Generators are easy to use. They can be called like regular functions and can be used in `for` loops.
* Versatility: Generators can be used to create sequences of values in a variety of ways. They can be used to generate sequences of numbers, strings, or any other type of object.

Generators are an essential tool for Python developers. By using generators, you can write more memory-efficient, readable, and versatile code.

48. Write a Python program to use an iterator.

An iterator is an object that can be iterated over, meaning that you can loop through its elements. Iterators are created using the `iter()` function. For example, the following code creates an iterator for the list `numbers`:

```python
numbers = [1, 2, 3, 4, 5]
iterator = iter(numbers)
```

You can then use the `next()` function to iterate over the elements in the iterator. For example, the following code uses the `next()` function to iterate over the elements in the `iterator` iterator:

```python
while True:
    try:
        element = next(iterator)
        print(element)
    except StopIteration:
        break
```

This code will print the numbers from 1 to 5.

Iterators are useful for looping through sequences of values in a memory-efficient way. For example, the following code uses an iterator to loop through the lines in a file:

```python
with open("myfile.txt", "r") as file:
    iterator = iter(file)
    while True:
        try:
            line = next(iterator)
            print(line)
        except StopIteration:
            break
```

This code will print the lines in the file `myfile.txt`.

Iterators are also used in `for` loops. For example, the following code uses a `for` loop to iterate over the elements in the `numbers` list:

```python
numbers = [1, 2, 3, 4, 5]
for number in numbers:
   print(number)
```

This code will print the numbers from 1 to 5.

Iterators are a powerful tool that can be used to loop through sequences of values in a memory-efficient way. They are also easy to use and can be used in a variety of ways.

Here are some of the benefits of using iterators:

*** Memory efficiency: Iterators are memory efficient because they only store the current value in the sequence. This means that they can be used to loop through sequences of values that are too large to store in memory all at once.**
*** Ease of use: Iterators are easy to use. They can be created using the `iter()` function and can be used in `for` loops.**
*** Versatility: Iterators can be used to loop through sequences of values in a variety of ways. They can be used to loop through lists, tuples, strings, files, and other types of objects.**

Iterators are an essential tool for Python developers. By using iterators, you can write more memory-efficient, readable, and versatile code.

49. Write a Python program to use a decorator.

A decorator is a function that takes another function as an argument and returns a new function. Decorators are used to modify the behavior of the function that they decorate. For example, the following decorator adds a `@logged` annotation to the function that it decorates:

```python
def logged(func):
   def wrapper(*args, kwargs):
      print(f"Calling function {func.__name__} with args {args} and kwargs
{kwargs}")
      result = func(*args, kwargs)
      print( f"Function {func.__name__} returned {result}")
      return result
   return wrapper
```

To use this decorator, you simply apply it to the function that you want to decorate. For example, the following code decorates the `greet()` function with the `@logged` decorator:

```python
@logged
def greet(name):
   print(f"Hello, {name}!")
```

Now, when you call the `greet()` function, the `@logged` decorator will be executed first. The `@logged` decorator will print a message to the console indicating that the `greet()` function is being called, and it will also print a message to the console indicating that the `greet()` function returned. For example, the following code calls the `greet()` function:

```python
greet("John Doe")
```

This code will print the following messages to the console:

```
Calling function greet with args ('John Doe',) and kwargs {}
Hello, John Doe!
Function greet returned None
```

```

Decorators are a powerful tool that can be used to modify the behavior of functions in a variety of ways. They are often used to add logging, error handling, or other functionality to functions.

Here are some of the benefits of using decorators:

* Code reusability: Decorators allow you to reuse code across multiple functions. This can help to reduce code duplication and make your code more maintainable.
* Modularity: Decorators allow you to modularize your code by separating the concerns of your functions from the concerns of the code that uses them. This can make your code easier to understand and maintain.
* Extensibility: Decorators allow you to extend the functionality of your functions without modifying the functions themselves. This can make it easy to add new features to your code without breaking existing  code.

Decorators are an essential tool for Python developers. By using decorators, you can write more reusable, modular, and extensible code.

Here are some examples of how decorators can be used:

* Logging: Decorators can be used to add logging to functions. This can be useful for debugging or for tracking the performance of your code.
* Error handling: Decorators can be used to handle errors in functions. This can help to make your code more robust and reliable.
* Caching: Decorators can be used to cache the results of functions. This can help to improve the performance of your code.
* Authorization: Decorators can be used to authorize users to access certain functions. This can help to protect your code from unauthorized access.

Decorators are a versatile tool that can be used to improve the quality and performance of your Python code.
```

50. Write a Python program to create a regular expression.

A regular expression is a sequence of characters that define a search pattern. Regular expressions are used to match text against a pattern. For example, the following regular expression matches any string that starts with the letter "a":

```
^a
```

To create a regular expression in Python, you can use the `re` module. The `re` module provides a number of functions and classes for working with regular expressions. For example, the following code creates a regular expression that matches any string that starts with the letter "a":

```python
 import re

pattern = re.compile("^a")
```

You can then use the `match()` method of the `re` module to match a regular expression against a string. For example, the following code uses the `match()` method to match the regular expression `^a` against the string "apple":

```python
import re

pattern = re.compile("^a")
match = pattern.match("apple")

if match:
   print("The string starts with the letter 'a'.")
else:
   print("The string does not start with the letter 'a'.")
```

This code will print the following message to the console:

```
The string starts with the letter 'a'.
```

Regular expressions are a powerful tool that can be used to search and manipulate text. They are used in a variety of applications, including text editors, search engines, and programming languages.

Here are some of the benefits of using regular expressions:

*** Power: Regular expressions are a powerful tool for searching and manipulating text. They can be used to find patterns in text, extract data from text, and validate input.**
*** Flexibility: Regular expressions are flexible and can be used to match a wide variety of patterns. This makes them a valuable tool for a variety of tasks.**
*** Efficiency: Regular expressions are efficient and can be used to search and manipulate large amounts of text quickly.**

Regular expressions are an essential tool for Python developers. By using regular expressions, you can write more powerful, flexible, and efficient code.

Here are some examples of how regular expressions can be used:

*** Text search: Regular expressions can be used to search text for specific patterns. This can be useful for finding files, extracting data from text, and performing other tasks.**
*** Data validation: Regular expressions can be used to validate input. This can help to ensure that users enter valid data into your programs.**
*** Text manipulation: Regular expressions can be used to manipulate text. This can be useful for formatting text, removing unwanted characters, and performing other tasks.**

Regular expressions are a versatile tool that can be used to improve the quality and performance of your Python code.

51. Write a Python program to match a pattern using a regular expression.

```python
import re

 Define the regular expression pattern
pattern = r "^[a-zA-Z0-9]{6,12}$"

 Compile the regular expression pattern into a regular expression object
regex =  re.compile(pattern)

 Define the input string to be matched
input_string = "Hello123"

 Use the regular expression object to match the input string
match = regex.match(input_string)

 Check if the input string matches the regular expression pattern
if match :
   print("The input string matches the regular expression pattern.")
else:
   print("The input string does not match the regular expression pattern.")
```

Output:

```
The input string matches the regular expression pattern.
```

52. Write a Python program to split a string using a regular expression.

```python
import re
```

 Define the regular expression pattern to split the string
pattern = r"[,]"

 Define the input string to be split
input_string = "Hello, world! This is a sample string ."

 Use the regular expression pattern to split the input string
split_string = re.split(pattern, input_string)

 Print the split string
print(split_string)
```

Output:

```
['Hello', 'world!', 'This', 'is', 'a',  'sample', 'string.']
~~~

**In this example, the regular expression pattern `r"[, ]"` matches either a comma (`,`) or a space (` `). The `re.split()` function uses this pattern to split the input string into a list of substrings, where each substring is separated by a comma or a space.**

## 53. Write a Python program to search a string using a regular expression.

```python
import re

 Define the regular expression pattern to search for
 pattern = r"world"

 Define the input string to be searched
input_string = "Hello, world! This is a sample string."
~~~

```
 Use the regular expression pattern to search the input string
match = re.search(pattern, input_string)

 Check if the pattern was found in the input string
if match:
     Print the starting and ending positions of the match
    print("The pattern was found at positions {} to  {}.".format(match.start(), match.end()))
else:
    print("The pattern was not found in the input string.")
```

Output:

```
The pattern was found at positions 7 to 12.
```

In this example, the regular expression pattern `r"world"` matches the substring "world" in the input string. The `re.search()` function uses this pattern to search for the first occurrence of the pattern in the input string. If the pattern is found, the `match` object contains information about the match, including the starting and ending positions of the match.

54. Write a Python program to create a thread.

```python
import threading

 Define a function to be executed by the thread
def thread_function():
    print("Hello from a thread!")

 Create a thread object
thread = threading.Thread(target=thread_ function)
```

 Start the thread
thread.start()

 Join the thread with the main thread
thread.join()

print("Main thread completed.")
```

**Output:**

```
Hello from a thread!
Main thread completed.

In this example, the `threading.Thread()` class is used to create a thread object. The `target` parameter specifies the function to be executed by the thread. The `start()` method is then called to start the thread. The `join()` method is used to wait for the thread to complete its execution before continuing with the main thread.

When the `thread_function()` is executed by the thread, it prints "Hello from a thread!". Once the thread has completed its execution, the main thread prints "Main thread completed.".

55. Write a Python program to create a process.

```python
import os

 Define a function to be executed by the child  process
def child_process():
   print("Hello from the child process!")
   os._exit(0)   Exit the child process
```

```
  Create a child process
pid = os.fork()

 Check if the current process is the child process
if pid == 0:
    child_process()
else:
     This is the parent process
    print("Hello from the parent process!")

print("Process completed.")
```

Output:

```
Hello from the parent process!
Hello from the child process!
Process completed.
```

In this example, the `os.fork()` function is used to create a child process. The `fork()` function returns 0 in the child process and the process ID of the child process in the parent process.

The `if pid == 0:` statement is used to check if the current process is the child process. If it is, the `child_process()` function is executed. Otherwise, the parent process continues execution.

The `os._exit(0)` function is used to exit the child process. This is necessary because the child process is created by copying the parent process, and if the child process does not exit, it will continue to run even after the parent process has completed.

The `print()` statements are used to display messages from the parent and child processes.

56. Write a Python program to create a socket.

```python
import socket

 Create a socket object
sock = socket. socket(socket.AF_INET, socket.SOCK_STREAM)

 Bind the socket to a local address and port
sock.bind(('localhost',  12345))

 Listen for incoming connections
sock.listen(1)

 Accept an incoming connection
conn, addr = sock.accept()

 Receive data from the client
data = conn.recv(1024)

 Send data back to the client
conn .send(data.upper())

 Close the connection
conn.close()

 Close the socket
sock.close()
```

In this example, the `socket.socket()` function is used to create a socket object. The `socket.AF_INET` parameter specifies that the socket will be used for IPv4 addresses, and the `socket.SOCK_STREAM` parameter specifies that the socket will be used for TCP connections.

The `sock.bind()` method is then used to bind the socket to a local address and

port. In this case, the socket is bound to the localhost address (12 7.0.0.1) and port 12345.

The `sock.listen()` method is then used to listen for incoming connections. The `1` parameter specifies that the socket can accept up to 1 incoming connection at a time.

The `sock.accept()` method is then used to accept an incoming connection. This method blocks until a connection is received. Once a connection is received, the `accept()` method returns a tuple containing the socket object for the connection and the address of the client.

The `conn.recv()` method is then used to receive data from the client. The `1024` parameter specifies the maximum number of bytes to receive.

The `conn.send()` method is then used to send data back to the client. The `data.upper()` expression converts the received data to uppercase before sending it back to the client.

Finally, the `conn.close()` method is used to close the connection, and the `sock.close()` method is used to close the socket.

57. Write a Python program to create a server socket and accept connections.

```python
import socket

 Create a server socket
server_socket =  socket.socket(socket.AF_INET, socket.SOCK_STREAM)

 Bind the server socket to a local address and port
server_socket .bind(('localhost', 12345))

 Listen for incoming connections
server_socket.listen(1)
```

```
 Accept an incoming connection
client_socket, client_address = server_socket.accept()

 Receive data from the client
data = client_socket.recv( 1024)

 Send data back to the client
client_socket.send(data.upper())

 Close the client socket
client_socket.close()

 Close the server socket
server_socket.close()
```

In this example, the `socket.socket()` function is used to create a server socket object. The `socket.AF_INET` parameter specifies that the socket will be used for IPv4 addresses, and the `socket.SOCK_STREAM` parameter specifies that the socket will be used for TCP connections.

The `server_socket.bind()` method is then used to bind the server socket to a local address and port. In this case, the server socket is bound to the localhost address (127.0.0.1) and port 12345.

The `server_socket.listen()` method is then used to listen for incoming connections. The `1` parameter specifies that the server socket can accept up to 1 incoming connection at a time.

The `server_socket.accept()` method is then used to accept an incoming connection. This method blocks until a connection is received. Once a connection is received, the `accept()` method returns a tuple containing the socket object for the connection and the address of the client.

The `client_socket.recv()` method is then used to receive data from the client. The `1024` parameter specifies the maximum number of bytes to receive.

The `client_socket.send()` method is then used to send data back to the client. The `data.upper()` expression converts the received data to uppercase before sending it back to the client.

Finally, the `client_socket.close()` method is used to close the connection, and the `server_socket.close()` method is used to close the server socket.

58. Write a Python program to create a client socket and send data.

```python
import socket

 Create a client socket
client_socket =  socket.socket(socket.AF_INET, socket.SOCK_STREAM)

 Connect the client socket to the server socket
client_socket.connect (('localhost', 12345))

 Send data to the server
client_socket.send(b'Hello from the client!')

 Receive data from the server
data = client_socket.recv(1024)

 Print the data received from the server
print( data.decode())

 Close the client socket
client_socket.close()
```

In this example, the `socket.socket()` function is used to create a client socket object. The `socket.AF_INET` parameter specifies that the socket will be used for IPv4 addresses, and the `socket.SOCK_STREAM` parameter specifies that the socket will be used for TCP connections.

The `client_socket.connect()` method is then used to connect the client socket to the server socket. The `('localhost', 12345)` tuple specifies the address and port of the server socket.

The `client_socket.send()` method is then used to send data to the server socket. The `b'Hello from the client!'` expression converts the string to bytes before sending it to the server socket.

The `client_socket.recv()` method is then used to receive data from the server socket. The `1024` parameter specifies the maximum number of bytes to receive.

The `data.decode()` method is then used to decode the received data from bytes to a string.

Finally, the `client_socket.close()` method is used to close the client socket.

59. Write a Python program to create a UDP socket.

```python
import socket

 Create a UDP socket
sock = socket. socket(socket.AF_INET, socket.SOCK_DGRAM)

 Bind the socket to a local address and port
sock.bind(('localhost ', 12345))

 Receive data from the client
data, addr = sock.recvfrom(1024)

 Send data back to the client
sock.sendto(data.upper(), addr)

 Close the socket
```

sock.close()

In this example, the `socket.socket()` function is used to create a UDP socket object. The `socket.AF_INET` parameter specifies that the socket will be used for IPv4 addresses, and the `socket.SOCK_DGRAM` parameter specifies that the socket will be used for UDP connections.

The `sock.bind()` method is then used to bind the socket to a local address and port. In this case, the socket is bound to the localhost address (127.0.0.1) and port 12345.

The `sock.recvfrom()` method is then used to receive data from the client. The `1024` parameter specifies the maximum number of bytes to receive. The `recvfrom()` method also returns the address of the client that sent the data.

The `sock.sendto()` method is then used to send data back to the client. The `data.upper()` expression converts the received data to uppercase before sending it back to the client. The `addr` parameter specifies the address of the client to send the data to.Finally, the `sock.close()` method is used to close the socket.

60. Write a Python program to create a GUI application using tkinter.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a label
label = tk.Label(text="Hello, world!")
 label.pack()

 Create a button
```

```
button = tk.Button(text="Click me!")
button.pack()

 Define a function to handle the button click
def button_click():
   label.config(text="Button clicked!")

 Bind the button click event to the button_ click function
button.config(command=button_click)

 Start the main event loop
window.mainloop()
```

This program creates a simple GUI application using the tkinter library. The application consists of a main window with a label and a button. When the button is clicked, the text of the label is changed to "Button clicked!".

To run the program, save the code in a file with a `.py` extension (for example, `gui.py`) and then run the following command in a terminal:

```
python gui.py
```

This will open the GUI application window. You can then click the button to change the text of the label.

61. Write a Python program to create a button in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")
```

```
 Create a button
button = tk.Button(text="Click me!")
button .pack()

 Define a function to handle the button click
def button_click():
   print("Button clicked!")

 Bind the button click event to the button_click function
button.config(command=button_click)

 Start the main event loop
window.mainloop()
```

In this program, a button is created using the `tk.Button()` constructor. The `text` parameter specifies the text to be displayed on the button. The `pack()` method is then used to add the button to the main window.The `button_click()` function is defined to handle the button click event. This function simply prints a message to the console. The `button.config(command=button_click)` line binds the button click event to the `button_click()` function.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly.

When the button is clicked, the `button_click()` function is called and the message "Button clicked!" is printed to the console.

62. Write a Python program to create a text box in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a text box
text_box = tk.Text()
text_box .pack()

 Define a function to handle the text box input
def text_box_input(event):
   print("Text box input:", text_box.get("1.0", "end"))

 Bind the text box input event to the text_box_input function
text_ box.bind("<KeyRelease>", text_box_input)

 Start the main event loop
window.mainloop()
```

In this program, a text box is created using the `tk.Text()` constructor. The `pack()` method is then used to add the text box to the main window.

The `text_box_input()` function is defined to handle the text box input event. This function simply prints the text that is currently in the text box to the console. The `text_box.get("1.0", "end")` line gets the text from the text box, starting from the first character in the first line and ending with the last character in the last line.

The `text_box.bind("<KeyRelease>", text_box_input)` line binds the text box input event to the `text_box_input()` function. This means that the `text_box_input()` function will be called every time a key is released in the text box.

Finally, the `window.mainloop()` method is called to start the main event

loop. This loop listens for events such as text box input and updates the GUI accordingly.

When you type text into the text box and release a key, the `text_box_input()` function will be called and the text that is currently in the text box will be printed to the console.
63. Write a Python program to create a menu in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a menu bar
menu_bar = tk.Menu(window)
window .config(menu=menu_bar)

 Create a menu item
file_menu = tk.Menu(menu_bar, tearoff=0)
file_menu.add_command(label="New")
file_menu.add_command(label="Open")
file_menu. add_command(label="Save")
file_menu.add_separator()
file_menu.add_command(label="Exit", command=window.quit)

 Add the menu item to the menu bar
menu_bar.add_cascade(label="File", menu=file_menu)

 Start the main event loop
window.mainloop()
```

In this program, a menu bar is created using the `tk.Menu()` constructor. The `window.config(menu=menu_bar)` line adds the menu bar to the main

window. A menu item is then created using the `tk.Menu()` constructor. The `tearoff=0` parameter specifies that the menu item cannot be torn off from the menu bar. The `add_command()` method is then used to add commands to the menu item. The `label` parameter specifies the text to be displayed on the menu item, and the `command` parameter specifies the function to be called when the menu item is clicked. Finally, the `add_cascade()` method is used to add the menu item to the menu bar. The `label` parameter specifies the text to be displayed on the menu bar, and the `menu` parameter specifies the menu item to be added. When the menu item is clicked, the `window.quit()` function is called, which closes the main window and exits the application.

You can also add submenus to the menu bar by creating a new `tk.Menu()` object for each submenu and then using the `add_cascade()` method to add the submenu to the menu bar.

64. Write a Python program to create a dialog box in a GUI application.

~~~
```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a function to handle the dialog box button click
def dialog_box_button _click():
   print("Dialog box button clicked!")

 Create a dialog box
dialog_box = tk.Toplevel(window)
dialog_box.title("Dialog Box")

 Create a label in the dialog box
label = tk.Label(dialog_box, text="This  is a dialog box.")
~~~

label.pack()

Create a button in the dialog box
button = tk.Button(dialog_box, text="Click me!",
command=dialog_box_button_click)
button.pack()

Start the main event loop
window.mainloop()
**```**

In this program, a dialog box is created using the `tk.Toplevel()` constructor. The `window` parameter specifies the parent window of the dialog box. The `title` parameter specifies the title of the dialog box.

A label and a button are then created in the dialog box using the ` tk.Label()` and `tk.Button()` constructors, respectively. The `pack()` method is used to add the label and button to the dialog box.

The `dialog_box_button_click()` function is defined to handle the button click event. This function simply prints a message to the console. The `button.config(command=dialog_box_button_click)` line binds the button click event to the `dialog_box_button_click()` function.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly.

When the button in the dialog box is clicked, the `dialog_box_button_click()` function is called and the message "Dialog box button clicked!" is printed to the console.

65. Write a Python program to create a canvas in a GUI application.

~~~
```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a canvas
canvas = tk.Canvas(window, width=400 , height=300)
canvas.pack()

 Create a rectangle on the canvas
rectangle = canvas.create_rectangle(100, 100, 200, 200, fill="red")

 Create a line on the canvas
line = canvas .create_line(0, 0, 400, 300, fill="blue")

 Create an oval on the canvas
oval = canvas.create_oval(150, 150, 250, 250, fill="green")

 Start the main event loop
window.mainloop()
```
~~~

In this program, a canvas is created using the `tk.Canvas()` constructor. The `width` and `height` parameters specify the width and height of the canvas, respectively. The `pack()` method is then used to add the canvas to the main window. Three shapes (a rectangle, a line, and an oval) are then created on the canvas using the `create_rectangle()`, `create_line()`, and `create_oval()` methods, respectively. The `fill` parameter specifies the color of the shape.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as mouse clicks and updates the GUI accordingly.

When you run this program, you will see a window with a red rectangle, a blue line, and a green oval drawn on the canvas.

66. Write a Python program to create a frame in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a frame
frame = tk.Frame(window)
frame.pack()

 Create a label in the frame
label = tk.Label(frame, text="This is a frame.")
label.pack()

 Create a button in the frame
button = tk.Button(frame, text="Click me!")
button.pack()

 Start the main event loop
 window.mainloop()
```

In this program, a frame is created using the `tk.Frame()` constructor. The `window` parameter specifies the parent window of the frame. The `pack()` method is then used to add the frame to the main window.

A label and a button are then created in the frame using the `tk.Label()` and `tk.Button()` constructors, respectively. The `pack()` method is used to add the label and button to the frame.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly.

When you run this program, you will see a window with a frame containing a label and a button.

67. Write a Python program to create a scrollbar in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a canvas
canvas = tk.Canvas(window)
canvas.pack()

 Create a scrollbar
scrollbar = tk.Scrollbar(window, orient="vertical", command=canvas.yview)
scrollbar.pack(side="right", fill="y")

 Configure the canvas to use the scrollbar
canvas.configure(yscrollcommand=scrollbar.set)

  Create some content for the canvas
for i in range(100):
    canvas.create_text(100, 20 * i, text=f"Line {i}")

 Start the main event loop
window.mainloop()
```

In this program, a canvas and a scrollbar are created using the `tk.Canvas()`

and `tk.Scrollbar()` constructors, respectively. The `pack()` method is used to add the canvas and scrollbar to the main window. The `orient` parameter of the scrollbar specifies that it should be oriented vertically. The `command` parameter specifies that the scrollbar should be linked to the canvas's vertical scrollbar.

The `configure()` method of the canvas is used to configure the canvas to use the scrollbar for vertical scrolling.

Some content is then created for the canvas using a `for` loop. The `create_text()` method is used to create text objects on the canvas.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as mouse clicks and updates the GUI accordingly.

When you run this program, you will see a window with a canvas containing a lot of text. You can use the scrollbar to scroll up and down through the text.

68. Write a Python program to create a radiobutton in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a label
label = tk.Label(text="Choose your favorite color:")
label.pack()

 Create a variable to store the selected color
selected_color = tk.StringVar()

 Create a radio button for each color
```

```
colors = ["Red", "Green", "Blue"]
for color in colors:
    radio_button = tk.Radiobutton(window,  text=color,
variable=selected_color, value=color)
    radio_button.pack()

 Create a function to handle the radio button selection
def radio_button_selected(event):
    print(f"Selected color: {selected_color.get()}")

 Bind the radio button selection event to the radio_button_selected function
for radio_button in radio_buttons:
    radio_button.bind("<ButtonRelease>", radio_button_selected)

 Start the main event loop
window.mainloop()
```

In this program, a label and a radio button for each color are created using the `tk.Label()` and `tk.Radiobutton()` constructors, respectively. The `pack()` method is used to add the label and radio buttons to the main window.

A `tk.StringVar()` object is used to store the selected color. The `variable` and `value` parameters of the radio buttons are used to link the radio buttons to the `selected_color` variable.

The `radio_button_selected()` function is defined to handle the radio button selection event. This function simply prints the selected color to the console.

The `bind()` method is used to bind the radio button selection event to the `radio_button_selected()` function.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as radio button clicks and updates the GUI accordingly.

When you run this program, you will see a window with a label and three radio buttons, one for each color. You can click on a radio button to select your favorite color. The selected color will be printed to the console.

69. Write a Python program to create a checkbox in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a label
label = tk.Label(text="Select your favorite colors:")
label.pack()

 Create a list of colors
colors = ["Red", "Green", "Blue", "Yellow", "Orange", "Purple"]

 Create a checkbox for each color
checkboxes = []
for color in colors:
    checkbox = tk.Checkbutton(window, text =color)
    checkbox.pack()
    checkboxes.append(checkbox)

 Create a function to handle the checkbox selection
def checkbox_selected(event):
    selected_colors = []
    for checkbox in checkboxes:
        if checkbox.var.get():
            selected_colors.append(checkbox["text"])
    print(f"Selected colors: {selected_colors}")

 Bind the checkbox selection event to the checkbox_selected function
```

```
for checkbox in checkboxes:
   checkbox.var.trace("w", checkbox_selected)

 Start the main event loop
 window.mainloop()
```

In this program, a label and a checkbox for each color are created using the `tk.Label()` and `tk.Checkbutton()` constructors, respectively. The `pack()` method is used to add the label and checkboxes to the main window.

A list of `tk.Checkbutton()` objects is used to store the checkboxes.

The `checkbox_selected()` function is defined to handle the checkbox selection event. This function gets the text of the selected checkboxes and prints it to the console.

The `trace()` method is used to bind the checkbox selection event to the `checkbox_selected()` function.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as checkbox clicks and updates the GUI accordingly. When you run this program, you will see a window with a label and six checkboxes, one for each color. You can click on the checkboxes to select your favorite colors. The selected colors will be printed to the console.

70. Write a Python program to create a slider in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a label
label = tk.Label(text="Volume:")
label. pack()
```

```
 Create a slider
slider = tk.Scale(window, from_=0, to=10, orient=tk.HORIZONTAL)
slider.pack()

 Create a function to handle the slider movement
def slider_moved(event):
   print(f"Volume: {slider .get()}")

 Bind the slider movement event to the slider_moved function
slider.bind("<ButtonRelease>", slider_moved)

 Start the main event loop
window.mainloop()
```

In this program, a label and a slider are created using the `tk.Label()` and `tk.Scale()` constructors, respectively. The `pack()` method is used to add the label and slider to the main window.

The `from_` and `to` parameters of the slider specify the minimum and maximum values of the slider, respectively. The `orient` parameter specifies that the slider should be oriented horizontally.

The `slider_moved()` function is defined to handle the slider movement event. This function gets the current value of the slider and prints it to the console.

The `bind()` method is used to bind the slider movement event to the `slider_moved()` function.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as slider movement and updates the GUI accordingly.

When you run this program, you will see a window with a label and a slider. You can move the slider to adjust the volume. The current volume will be printed to the console.

71. Write a Python program to create a listbox in a GUI application.

~~~
```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a label
label = tk.Label(text="Select your favorite colors:")
label.pack()

 Create a listbox
listbox = tk.Listbox(window, selectmode=tk.MULTIPLE)
colors = ["Red", "Green", "Blue", "Yellow", "Orange", "Purple"]
for color in colors:
   listbox.insert(tk.END , color)
listbox.pack()

 Create a function to handle the listbox selection
def listbox_selected(event):
   selected_colors = [listbox.get(index) for index in listbox.curselection()]
   print(f"Selected colors: {selected_colors}")

 Bind the listbox selection event to the listbox_selected function
listbox.bind("<<ListboxSelect>>", listbox_selected)

 Start the main event loop
window.mainloop()
```
~~~

In this program, a label and a listbox are created using the `tk.Label ()` and `tk.Listbox()` constructors, respectively. The `pack()` method is used to add the label and listbox to the main window.

The `selectmode` parameter of the listbox is set to `tk.MULTIPLE`, which allows multiple items to be selected at the same time.

A list of colors is created and each color is inserted into the listbox using the `insert()` method.

The `listbox_selected()` function is defined to handle the listbox selection event. This function gets the text of the selected items and prints it to the console.

The `bind()` method is used to bind the listbox selection event to the `listbox_selected()` function.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as listbox selection and updates the GUI accordingly.

When you run this program, you will see a window with a label and a listbox containing a list of colors. You can click on the items in the listbox to select them. The selected colors will be printed to the console.

72. Write a Python program to create a spinbox in a GUI application.

~~~
```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a label
label = tk.Label(text="Select a number:")
 label.pack()

 Create a spinbox
~~~

```
spinbox = tk.Spinbox(window, from_=0, to=10)
spinbox.pack()

 Create a function to handle the spinbox value change
def spinbox_value_changed(event):
   print(f "Selected number: {spinbox.get()}")

 Bind the spinbox value change event to the spinbox_value_changed function
spinbox.bind("<<SpinboxValueChange>>", spinbox_value_changed)

 Start the main event loop
window.mainloop()
```

In this program, a label and a spinbox are created using the `tk.Label()` and `tk.Spinbox()` constructors, respectively. The `pack()` method is used to add the label and spinbox to the main window.

The `from_` and `to` parameters of the spinbox specify the minimum and maximum values of the spinbox, respectively.

The `spinbox_value_changed()` function is defined to handle the spinbox value change event. This function gets the current value of the spinbox and prints it to the console.

The `bind()` method is used to bind the spinbox value change event to the `spinbox_value_changed()` function.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as spinbox value change and updates the GUI accordingly.

When you run this program, you will see a window with a label and a spinbox. You can use the spinbox to select a number between 0 and 10. The selected number will be printed to the console.

73. Write a Python program to create a progressbar in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a label
label = tk.Label(text="Progress:")
label. pack()

 Create a progressbar
progressbar = tk.ttk.Progressbar(window, orient=tk.HORIZONTAL,
length=200, mode="determinate")
progressbar.pack()

 Create a function to update the progressbar
def update_progressbar():
    progressbar["value "] += 10
    if progressbar["value"] >= 100:
        progressbar["value"] = 0
    window.after(100, update_progressbar)

 Start the progressbar update
window.after(100, update_progressbar)

 Start the main event loop
window.mainloop()
```

In this program, a label and a progressbar are created using the `tk.Label()` and `tk.ttk.Progressbar()` constructors, respectively. The `pack()` method is used to add the label and progressbar to the main window .

The `orient` parameter of the progressbar specifies that it should be oriented horizontally. The `length` parameter specifies the length of the progressbar in pixels. The `mode` parameter specifies that the progressbar should be in determinate mode, which means that the progress can be accurately determined.

The `update_progressbar()` function is defined to update the progressbar. This function increments the progressbar's value by 10 and checks if the progressbar's value has reached 100. If it has, the progressbar's value is reset to 0. The `window.after()` method is used to call the `update_progressbar()` function every 100 milliseconds.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as progressbar updates and updates the GUI accordingly.

When you run this program, you will see a window with a label and a progressbar. The progressbar will start to fill up and will reset to 0 once it reaches 100.

74. Write a Python program to create a treeview in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a treeview
treeview = tk.ttk.Treeview(window)
treeview.pack()

 Define the columns
columns = ("Name", "Age", "Occupation")
```

```
 Insert the columns
for column in columns:
   treeview.heading(column, text=column)

 Insert some data
data = [
   ("John Doe", 30,  "Software Engineer"),
   ("Jane Smith", 25, "Doctor"),
   ("Bill Jones", 40, "Lawyer"),
]

for item in data:
   treeview.insert("", tk.END, values=item)

 Start the main event loop
window.mainloop()
```

In this program, a treeview is created using the `tk.ttk.Treeview()` constructor. The `pack()` method is used to add the treeview to the main window.

The `columns` tuple defines the columns of the treeview. The `heading()` method is used to insert the columns into the treeview.

The `data` list contains the data that will be inserted into the treeview. The `insert()` method is used to insert the data into the treeview.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as treeview item selection and updates the GUI accordingly. When you run this program, you will see a window with a treeview containing the data from the `data` list. You can click on the column headers to sort the data. You can also click on the items in the treeview to select them.

75. Write a Python program to create a tooltip in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a label
label = tk.Label(text="Hover over me for a  tooltip!")
label.pack()

 Create a tooltip
tooltip = tk.ttk.Balloon(label)
tooltip.bind_widget(label, balloonmsg="This is a tooltip!")

 Start the main event loop
window.mainloop()
```

In this program, a label and a tooltip are created using the `tk.Label()` and `tk.ttk.Balloon()` constructors, respectively. The `pack()` method is used to add the label to the main window.

The `bind_widget()` method is used to bind the tooltip to the label. The `balloonmsg` parameter specifies the text that will be displayed in the tooltip.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as mouse movement and updates the GUI accordingly. When you run this program, you will see a window with a label. If you hover your mouse over the label, a tooltip will appear.

76. Write a Python program to create a date picker in a GUI application.

```python
import tkinter as tk
from tkcalendar import DateEntry

  Create the main window
window = tk.Tk()
window.title("My GUI Application")

 Create a label
label = tk.Label( text="Select a date:")
label.pack()

 Create a date picker
date_picker = DateEntry(window, width=12)
date_picker.pack()

 Start the main event loop
window.mainloop()
```

In this program, a label and a date picker are created using the `tk.Label()` and `tkcalendar.DateEntry()` constructors, respectively. The `pack()` method is used to add the label and date picker to the main window.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as date selection and updates the GUI accordingly. When you run this program, you will see a window with a label and a date picker. You can click on the date picker to select a date. The selected date will be displayed in the date picker.

77. Write a Python program to create a time picker in a GUI application.

```python
import tkinter as tk
from tkcalendar import TimeEntry

  Create the main window
window = tk.Tk()
window.title("My GUI Application")

 Create a label
label = tk.Label( text="Select a time:")
label.pack()

 Create a time picker
time_picker = TimeEntry(window, width=12)
time_picker.pack()

 Start the main event loop
window.mainloop()
```

In this program, a label and a time picker are created using the `tk.Label()` and `tkcalendar.TimeEntry()` constructors, respectively. The `pack()` method is used to add the label and time picker to the main window.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as time selection and updates the GUI accordingly.

When you run this program, you will see a window with a label and a time picker. You can click on the time picker to select a time. The selected time will be displayed in the time picker.

78. Write a Python program to create a color picker in a GUI application.

~~~
```python
import tkinter as tk
from tkinter import colorchooser

 Create  the main window
window = tk.Tk()
window.title("My GUI Application")

 Create a label
label = tk.Label(text ="Select a color:")
label.pack()

 Create a color picker
def choose_color():
   color = colorchooser.askcolor()
   label.config(bg=color[1])

button = tk.Button(text="Choose Color", command=choose_color)
button .pack()

 Start the main event loop
window.mainloop()
```
~~~

In this program, a label and a button are created using the `tk.Label()` and `tk.Button()` constructors, respectively. The `pack()` method is used to add the label and button to the main window.

The `choose_color()` function is defined to handle the button click event. This function uses the `colorchooser.askcolor()` function to open a color picker dialog. The `colorchooser.askcolor()` function returns a tuple containing the RGB values of the selected color and the hexadecimal representation of the color.

The `label.config(bg=color[1])` line sets the background color of the label to

the selected color.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly.

When you run this program, you will see a window with a label and a button. You can click on the button to open a color picker dialog. Once you have selected a color, the background color of the label will change to the selected color.

79. Write a Python program to create a file dialog in a GUI application.

```python
import tkinter as tk
from tkinter import filedialog

 Create  the main window
window = tk.Tk()
window.title("My GUI Application")

 Create a label
label = tk.Label(text ="Select a file:")
label.pack()

 Create a file dialog
def choose_file():
   filename = filedialog.askopenfilename()
   label.config(text=filename)

button = tk.Button(text="Choose File", command=choose_file)
button. pack()

 Start the main event loop
window.mainloop()
```

In this program, a label and a button are created using the `tk.Label()` and `tk.Button()` constructors, respectively. The `pack()` method is used to add the label and button to the main window.

The `choose_file()` function is defined to handle the button click event. This function uses the `filedialog.askopenfilename()` function to open a file dialog. The `filedialog.askopenfilename()` function returns the path to the selected file.

The `label.config(text=filename)` line sets the text of the label to the path of the selected file. Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly. When you run this program, you will see a window with a label and a button. You can click on the button to open a file dialog. Once you have selected a file, the path to the selected file will be displayed in the label.

80. Write a Python program to create a message box in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a function to handle the button click event
def button_click():
    tk.messagebox.showinfo(title="Message", message="Hello, world!")
 Create a button
button = tk.Button(text="Click me!", command=button_click)
button.pack()
 Start the main event loop
window.mainloop()
```

In this program , a button is created using the `tk.Button()` constructor. The `pack()` method is used to add the button to the main window.

The `button_click()` function is defined to handle the button click event. This function uses the `tk.messagebox.showinfo()` function to display a message box. The `title` parameter specifies the title of the message box, and the `message` parameter specifies the message to be displayed in the message box.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly. When you run this program, you will see a window with a button. When you click on the button, a message box will appear with the title "Message" and the message "Hello, world!".

81. Write a Python program to create a table in a GUI application.

```python
import tkinter as tk
from tkinter import ttk

 Create the  main window
window = tk.Tk()
window.title("My GUI Application")

 Create a table
table = ttk.Treeview(window )

 Define the columns
columns = ("Name", "Age", "Occupation")

 Insert the columns
for column in columns:
   table.heading(column, text=column)

 Insert some data
data = [
   ("John Doe", 30, "Software Engineer"),
   ("Jane Smith", 25, "Doctor"),
   ("Bill Jones", 40, "Lawyer"),
]

for item in data:
   table.insert("", tk.END, values=item)

 Pack the table
table.pack()

 Start the main event loop
window.mainloop()
```

In this program, a table is created using the `ttk.Treeview()` constructor. The `pack()` method is used to add the table to the main window.

The `columns` tuple defines the columns of the table. The `heading()` method is used to insert the columns into the table.

The `data` list contains the data that will be inserted into the table. The `insert()` method is used to insert the data into the table.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as table item selection and updates the GUI accordingly. When you run this program, you will see a window with a table containing the data from the `data` list. You can click on the column headers to sort the data. You can also click on the items in the table to select them.

82. Write a Python program to create a toolbar in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a toolbar
toolbar = tk.Frame(window)
toolbar.pack( side=tk.TOP, fill=tk.X)

 Create a button on the toolbar
button = tk.Button(toolbar, text="New")
button.pack(side=tk.LEFT, padx=2, pady=2)

 Create another button on the toolbar
button = tk .Button(toolbar, text="Open")
button.pack(side=tk.LEFT, padx=2, pady=2)

 Create a separator on the toolbar
separator = tk.ttk.Separator(toolbar, orient=tk.VERTICAL)
separator.pack(side=tk.LEFT, padx=2, pady=2)

 Create a drop-down menu on the toolbar
menu = tk.OptionMenu(toolbar, None, "File", "Edit", "View")
menu.pack(side=tk.LEFT, padx=2, pady=2)

 Start the  main event loop
window.mainloop()
```

In this program, a toolbar is created using the `tk.Frame()` constructor. The `pack()` method is used to add the toolbar to the main window.

Two buttons are created on the toolbar using the `tk.Button()` constructor. The `pack()` method is used to add the buttons to the toolbar.

A separator is created on the toolbar using the `tk.ttk.Separator()` constructor. The `pack()` method is used to add the separator to the toolbar.

A drop-down menu is created on the toolbar using the `tk.OptionMenu()` constructor. The `pack()` method is used to add the drop-down menu to the toolbar.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly. When you run this program, you will see a window with a toolbar containing two buttons, a separator, and a drop-down menu. You can click on the buttons and select items from the drop-down menu to perform various actions.

83. Write a Python program to create a status bar in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a status bar
status_bar = tk.Label(window, text=" Ready", bd=1, relief=tk.SUNKEN,
anchor=tk.W)
status_bar.pack(side=tk.BOTTOM, fill=tk.X)

 Create a function to update the status bar
def update_status_bar(text):
   status_bar. config(text=text)
 Start the main event loop
window.mainloop()
```

In this program, a status bar is created using the `tk.Label()` constructor. The `pack()` method is used to add the status bar to the main window.

The `update_status_bar()` function is defined to update the text of the status bar.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly.

When you run this program, you will see a window with a status bar at the bottom. The status bar will display the text "Ready". You can use the `update_status_bar()` function to update the text of the status bar.

For example, you could add the following code to the program to update the status bar when the user clicks on a button:

```python
def button_click():
    update_status_bar("Button clicked!")

button = tk.Button(window, text="Click me!", command=button_click)
button.pack()
```

When the user clicks on the button, the `button_click()` function will be called and the text of the status bar will be updated to "Button clicked!".

84. Write a Python program to create a menu bar in a GUI application.

```python
import tkinter as tk
```

```
 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a menu bar
menu_bar = tk.Menu(window)
window .config(menu=menu_bar)

 Create a menu item
file_menu = tk.Menu(menu_bar, tearoff=0)
file_menu.add_command(label="New")
file_menu.add_command(label="Open")
file_menu. add_command(label="Save")
file_menu.add_separator()
file_menu.add_command(label="Exit", command=window.quit)

 Add the menu item to the menu bar
menu_bar.add_cascade(label="File", menu=file_menu)

 Start the main event loop
window.mainloop()
```

In this program, a menu bar is created using the `tk.Menu()` constructor. The `window.config(menu=menu_bar)` line adds the menu bar to the main window.

A menu item is created using the `tk.Menu()` constructor. The `tearoff=0` parameter specifies that the menu item cannot be torn off from the menu bar. The `add_command()` method is used to add commands to the menu item. The `label` parameter specifies the text to be displayed on the menu item, and the `command` parameter specifies the function to be called when the menu item is clicked.

Finally, the `add_cascade()` method is used to add the menu item to the menu bar. The `label` parameter specifies the text to be displayed on the menu bar,

and the `menu` parameter specifies the menu item to be added.

When you run this program, you will see a window with a menu bar containing a "File" menu. The "File" menu contains four menu items: "New", "Open", "Save", and "Exit". You can click on the menu items to perform various actions. You can also add submenus to the menu bar by creating a new `tk.Menu()` object for each submenu and then using the `add_cascade()` method to add the submenu to the menu bar.

85. Write a Python program to create a notebook in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a notebook
notebook = tk.ttk.Notebook(window)
notebook .pack()

 Create two frames for the notebook
frame1 = tk.Frame(notebook)
frame2 = tk.Frame(notebook)

 Add the frames to the notebook
notebook.add(frame1, text="Page 1")
notebook.add(frame2, text="Page  2")

 Create a label in the first frame
label1 = tk.Label(frame1, text="This is page 1")
label1.pack()

 Create a label in the second frame
label2 = tk.Label(frame2, text="This is page 2")
```

label2.pack()

Start the main event loop
window.mainloop()
**```**

In this program, a notebook is created using the `tk.ttk.Notebook()` constructor. The `pack()` method is used to add the notebook to the main window.

Two frames are created for the notebook using the `tk.Frame()` constructor. The `add()` method is used to add the frames to the notebook. The `text` parameter specifies the text to be displayed on the tab for the frame. A label is created in each frame using the `tk.Label()` constructor. The `pack()` method is used to add the labels to the frames.
Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as tab changes and updates the GUI accordingly.

When you run this program, you will see a window with a notebook containing two tabs. The first tab is labeled "Page 1" and the second tab is labeled "Page 2". You can click on the tabs to switch between the pages. Each page contains a label with some text.

86. Write a Python program to create a paned window in a GUI application.

**```python**
import tkinter as tk

Create the main window
window = tk.Tk()
window.title("My GUI Application")

Create a paned window
paned_window = tk.PanedWindow(window, orient=tk.HORIZONTAL)

```
paned_window.pack(fill=tk.BOTH, expand=True)

 Create two frames for the paned window
frame1 = tk.Frame(paned_window)
frame2 = tk.Frame(paned_window)

  Add the frames to the paned window
paned_window.add(frame1, weight=1)
paned_window.add(frame2, weight=2)

 Create a label in the first frame
label1 = tk.Label(frame1, text="This is frame 1")
label1.pack()

 Create a label in the second frame
label2 = tk.Label(frame2, text="This is frame 2")
label2.pack()

 Start the main event loop
window.mainloop()
```

In this program, a paned window is created using the `tk.PanedWindow()` constructor. The `orient` parameter specifies that the paned window should be oriented horizontally. The `pack()` method is used to add the paned window to the main window. The `fill` and `expand` parameters specify that the paned window should fill the entire window and that it should expand to fill any extra space. Two frames are created for the paned window using the `tk.Frame()` constructor. The `add()` method is used to add the frames to the paned window. The `weight` parameter specifies the relative size of the frames. In this case, the second frame is twice as large as the first frame.

A label is created in each frame using the `tk.Label()` constructor. The `pack()` method is used to add the labels to the frames.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as window resizing and updates the

GUI accordingly. When you run this program, you will see a window with a paned window containing two frames. The first frame is labeled "This is frame 1" and the second frame is labeled "This is frame 2". You can drag the border between the frames to resize them.

87. Write a Python program to create a Toplevel window in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a function to open a Toplevel window
def open_toplevel_ window():
     Create a Toplevel window
    toplevel_window = tk.Toplevel(window)
    toplevel_window.title("Toplevel Window")

     Create a label in the Toplevel window
    label = tk.Label(toplevel_window , text="This is a Toplevel window")
    label.pack()

     Create a button in the Toplevel window to close it
    button = tk.Button(toplevel_window, text="Close",
command=toplevel_window.destroy)
    button.pack()
 Create a button in the main window to open the Toplevel window
button = tk.Button(window, text="Open Toplevel Window",
command=open_toplevel_window)
button.pack()
 Start the main event loop
window.mainloop()
```

In this program, a function is created to open a Toplevel window. The \`tk.Toplevel()\` constructor is used to create the Toplevel window. The \`title()\` method is used to set the title of the Toplevel window. A label is created in the Toplevel window using the \`tk.Label()\` constructor. The \`pack()\` method is used to add the label to the Toplevel window.

A button is created in the Toplevel window to close it. The \`command\` parameter of the button is set to \`toplevel_window.destroy\`. This means that when the button is clicked, the \`destroy()\` method of the Toplevel window will be called, which will close the window.

A button is created in the main window to open the Toplevel window. The \`command\` parameter of the button is set to \`open_toplevel_window\`. This means that when the button is clicked, the \`open_toplevel_window()\` function will be called, which will open the Toplevel window.

Finally, the \`window.mainloop()\` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly. When you run this program, you will see a window with a button labeled "Open Toplevel Window". When you click on the button, a Toplevel window will appear. The Toplevel window will contain a label with the text "This is a Toplevel window" and a button labeled "Close". When you click on the "Close" button, the Toplevel window will close.

88. Write a Python program to create a splash screen in a GUI application.

```python
import tkinter as tk
import time

 Create the main window
window = tk.Tk()
window.title("My GUI Application")
```

```
 Create a splash screen
splash_screen = tk.Toplevel (window)
splash_screen.title("Splash Screen")
splash_screen.geometry("300x200")
splash_screen.configure(bg="white")

 Create a label in the splash screen
label = tk.Label(splash_screen, text="Loading...", font =("Arial", 20))
label.pack()

 Create a progress bar in the splash screen
progress_bar = tk.ttk.Progressbar(splash_screen, orient=tk.HORIZONTAL,
length=200, mode="indeterminate")
progress_bar.pack()

 Start the progress bar
progress_bar.start()

 Simulate loading data
time.sleep(5)

 Destroy the splash screen
splash_screen.destroy()

 Show the main window
window.deiconify()

 Start the main event loop
window.mainloop()
```

In this program, a splash screen is created using the `tk.Toplevel()` constructor. The `title()` method is used to set the title of the splash screen. The `geometry()` method is used to set the size of the splash screen. The `configure()` method is used to set the background color of the splash screen.

A label is created in the splash screen using the `tk.Label()` constructor. The

`text` and `font` parameters are used to set the text and font of the label, respectively. The `pack()` method is used to add the label to the splash screen.

A progress bar is created in the splash screen using the `tk.ttk.Progressbar()` constructor. The `orient` parameter specifies that the progress bar should be oriented horizontally. The `length` parameter specifies the length of the progress bar in pixels. The `mode` parameter specifies that the progress bar should be in indeterminate mode, which means that the progress cannot be accurately determined. The `pack()` method is used to add the progress bar to the splash screen.

The `start()` method of the progress bar is called to start the progress bar.

The `time.sleep()` function is used to simulate loading data .

The `destroy()` method of the splash screen is called to destroy the splash screen.

The `deiconify()` method of the main window is called to show the main window.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly. When you run this program, you will see a splash screen with a label that says "Loading..." and a progress bar. After 5 seconds, the splash screen will disappear and the main window will appear.

89. Write a Python program to create a print dialog in a GUI application.

```python
import tkinter as tk
from tkinter import filedialog

 Create  the main window
window = tk.Tk()
window.title("My GUI Application")
```

```
 Create a function to open a print dialog
def print _dialog():
    Get the filename of the file to be printed
   filename = filedialog.askopenfilename()

    If a filename was selected, open a print dialog
   if filename:
      print_dialog = tk.PrintDialog()

       Get the printer settings from the  print dialog
      printer_settings = print_dialog.show()

       If the user clicked "OK", print the file
      if printer_settings:
         print("Printing file:", filename)

 Create a button in the main window to open the print dialog
button = tk.Button(window, text="Print", command=print_dialog)
button.pack()

 Start the main event loop
window.mainloop()
```

In this program, a function is created to open a print dialog. The `filedialog.askopenfilename()` function is used to get the filename of the file to be printed.

If a filename was selected, a print dialog is opened using the `tk.PrintDialog()` constructor. The `show()` method of the print dialog is called to display the print dialog.

If the user clicked "OK" in the print dialog, the printer settings are retrieved using the `printer_settings` attribute of the print dialog. The `print()` function is then used to print the file.

A button is created in the main window to open the print dialog. The `command` parameter of the button is set to `print_dialog`. This means that when the button is clicked, the `print_dialog()` function will be called.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly. When you run this program, you will see a window with a button labeled "Print". When you click on the button, a print dialog will appear. You can select the printer and other print settings in the print dialog. If you click "OK", the file will be printed.

90. Write a Python program to create a font chooser in a GUI application.

```python
import tkinter as tk
from tkinter import font

 Create the  main window
window = tk.Tk()
window.title("My GUI Application")

 Create a function to open a font chooser
def choose _font():
     Create a font chooser dialog
    font_chooser = tk.font.Font(family="Arial", size=12)

     Get the font from the font chooser dialog
    font_chooser.configure(font_chooser.askfont())

     Set  the font of the label to the selected font
    label.config(font=font_chooser)

 Create a label in the main window
label = tk.Label(text="Hello, world!")
label.pack()
```

```
 Create a button in the main window to open the font chooser
button = tk.Button(text="Choose Font", command=choose_font)
button.pack()

 Start the main event loop
window.mainloop()
```

In this program, a function is created to open a font chooser. The `tk.font.Font()` constructor is used to create a font chooser dialog. The `askfont()` method of the font chooser dialog is called to display the font chooser dialog.

If the user selected a font in the font chooser dialog, the font is retrieved using the `configure()` method of the font chooser dialog. The `font` attribute of the label is then set to the selected font.

A label is created in the main window to display the text.

A button is created in the main window to open the font chooser. The `command` parameter of the button is set to `choose_font`. This means that when the button is clicked, the `choose_font()` function will be called.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly.   When you run this program, you will see a window with a label that says "Hello, world!" and a button labeled "Choose Font". When you click on the button, a font chooser dialog will appear. You can select a font and other font settings in the font chooser dialog. If you click "OK", the font of the label will be changed to the selected font.

91. Write a Python program to create a layout manager in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a frame to hold the widgets
frame = tk.Frame(window)
 frame.pack()

 Create a label in the frame
label = tk.Label(frame, text="Hello, world!")
label.grid(row=0, column=0)

 Create a button in the frame
button = tk.Button(frame, text="Click me!")
button .grid(row=0, column=1)

 Start the main event loop
window.mainloop()
```

In this program, a frame is created using the `tk.Frame()` constructor. The `pack()` method is used to add the frame to the main window.

A label is created in the frame using the `tk.Label()` constructor. The `grid()` method is used to add the label to the frame. The `row` and `column` parameters specify the position of the label in the grid.

A button is created in the frame using the `tk.Button()` constructor. The `grid()` method is used to add the button to the frame. The `row` and `column` parameters specify the position of the button in the grid.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly.

When you run this program, you will see a window with a frame containing a label and a button. The label is positioned in the first row and first column of the grid, and the button is positioned in the first row and second column of the grid. You can use the `grid()` method to position widgets in a grid layout. The `row` and `column` parameters specify the position of the widget in the grid. You can also use the `sticky` parameter to specify how the widget should be aligned within its cell in the grid.
For more information on grid layouts, you can refer to the Tkinter

documentation: https://www.tcl.tk/man/tcl8.6/TkCmd/grid.htm

92. Write a Python program to create a grid layout in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a frame to hold the widgets
frame = tk.Frame(window)
 frame.pack()

 Create a grid layout manager for the frame
frame.columnconfigure(0, weight=1)
frame.columnconfigure(1, weight=1)
frame.rowconfigure(0, weight=1)
frame.rowconfigure(1, weight=1)
```

```
  Create a label in the frame
label = tk.Label(frame, text="Hello, world!")
label.grid(row=0, column=0, sticky="nsew")

 Create a button in the frame
button = tk.Button(frame, text="Click me!")
button.grid(row=0, column=1, sticky="nsew")

 Create another label in the frame
label2 = tk.Label(frame, text="This is another label")
label2.grid(row=1, column=0, sticky="nsew")

 Create another button  in the frame
button2 = tk.Button(frame, text="Click me too!")
button2.grid(row=1, column=1, sticky="nsew")

 Start the main event loop
window.mainloop()
```

In this program, a grid layout manager is created for the frame using the `columnconfigure()` and `rowconfigure()` methods. The `weight` parameter specifies that the columns and rows should be equally sized.
Four widgets are created in the frame: two labels and two buttons. The `grid()` method is used to add the widgets to the frame. The `row` and `column` parameters specify the position of the widget in the grid. The `sticky` parameter specifies how the widget should be aligned within its cell in the grid. In this case, the widgets are aligned to the north, south, east, and west sides of their cells.
Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly. When you run this program, you will see a window with a frame containing four widgets: two labels and two buttons. The widgets are arranged in a grid layout, with the labels on the left and the buttons on the right. The widgets are equally sized and they are aligned to the north, south, east, and west sides of their cells.

93. Write a Python program to create a pack layout in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a frame to hold the widgets
frame = tk.Frame(window)
 frame.pack()

 Create a label in the frame
label = tk.Label(frame, text="Hello, world!")
label.pack()

 Create a button in the frame
button = tk.Button(frame, text="Click me!")
button.pack()

 Start the main  event loop
window.mainloop()
```

In this program, a frame is created using the `tk.Frame()` constructor. The `pack()` method is used to add the frame to the main window.

A label is created in the frame using the `tk.Label()` constructor. The `pack()` method is used to add the label to the frame.

A button is created in the frame using the `tk.Button()` constructor. The `pack()` method is used to add the button to the frame.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly. When you run this program, you will see a window with a frame containing a label and a button. The label and button are stacked vertically within the frame.

You can use the `pack()` method to arrange widgets in a pack layout. The `pack()` method automatically arranges the widgets in the order they are added to the frame. You can use the `side` parameter to specify whether the widget should be packed on the left, right, top, or bottom of the frame.

For more information on pack layouts, you can refer to the Tkinter documentation: https://www.tcl.tk/man/tcl8.6/TkCmd/pack.htm

94. Write a Python program to create a place layout in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("My GUI Application")

 Create a frame to hold the widgets
frame = tk.Frame(window)
 frame.pack()

 Create a label in the frame
label = tk.Label(frame, text="Hello, world!")
label.place(x=10, y=10)

 Create a button in the frame
button = tk.Button(frame, text="Click me!")
button.place(x=100, y=100)
```

```
 Start the main event loop
window.mainloop()
```

In this program, a frame is created using the `tk.Frame()` constructor. The `pack()` method is used to add the frame to the main window.

A label is created in the frame using the `tk.Label()` constructor. The `place()` method is used to add the label to the frame. The `x` and `y` parameters specify the position of the label in the frame.

A button is created in the frame using the `tk .Button()` constructor. The `place()` method is used to add the button to the frame. The `x` and `y` parameters specify the position of the button in the frame.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly.

When you run this program, you will see a window with a frame containing a label and a button. The label is positioned at the coordinates (10, 10) within the frame, and the button is positioned at the coordinates (100, 100) within the frame. You can use the `place()` method to position widgets at specific coordinates within a frame. The `x` and `y` parameters specify the coordinates of the top-left corner of the widget. For more information on place layouts, you can refer to the Tkinter documentation:

https://www.tcl.tk/man/tcl8.6/TkCmd/place.htm

95. Write a Python program to create a form in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
```

```
window.title("My GUI Application")

 Create a frame to hold the form
frame = tk.Frame(window)
 frame.pack()

 Create a label for the name field
label_name = tk.Label(frame, text="Name:")
label_name.grid(row=0, column=0)

 Create an entry field for the name
entry_name = tk.Entry(frame)
 entry_name.grid(row=0, column=1)

 Create a label for the email field
label_email = tk.Label(frame, text="Email:")
label_email.grid(row=1, column=0)

 Create an entry field for the email
entry_email = tk.Entry(frame)
entry_email.grid(row=1, column=1)

 Create a button to submit the form
button_submit = tk.Button(frame, text="Submit")
button_submit.grid(row=2, column=1)

  Start the main event loop
window.mainloop()
```

In this program, a frame is created using the `tk.Frame()` constructor. The `pack()` method is used to add the frame to the main window.

Two labels are created in the frame using the `tk.Label()` constructor. The `grid()` method is used to add the labels to the frame. The `row` and `column` parameters specify the position of the labels in the grid.

Two entry fields are created in the frame using the `tk.Entry()` constructor. The `grid()` method is used to add the entry fields to the frame. The `row` and `column` parameters specify the position of the entry fields in the grid.

A button is created in the frame using the `tk.Button()` constructor. The `grid()` method is used to add the button to the frame. The `row` and `column` parameters specify the position of the button in the grid.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly. When you run this program, you will see a window with a form containing two labels, two entry fields, and a button. The labels are for the name and email fields, and the entry fields are where the user can enter their name and email address. The button is used to submit the form.

You can use the `grid()` method to arrange widgets in a grid layout. The `row` and `column` parameters specify the position of the widget in the grid. You can also use the `sticky` parameter to specify how the widget should be aligned within its cell in the grid. For more information on grid layouts, you can refer to the Tkinter documentation:
https://www.tcl.tk/man/tcl8.6/TkCmd/grid.htm

96. Write a Python program to create a login form in a GUI application.

```python
import tkinter as tk
 Create the main window
window =  tk.Tk()
window.title("Login Form")
 Create a frame to hold the form
frame = tk.Frame(window)
frame .pack()
 Create a label for the username field
label_username = tk.Label(frame, text="Username:")
label_username.grid(row=0, column=0)
```

```
 Create an entry field for the username
entry_username = tk.Entry(frame)
entry _username.grid(row=0, column=1)

 Create a label for the password field
label_password = tk.Label(frame, text="Password:")
label_password.grid(row=1, column=0)

 Create an entry field for the password
entry_password = tk.Entry(frame, show="*")
entry_password.grid(row=1, column=1)

 Create a button to submit the form
button_submit = tk.Button(frame, text="Login")
button_submit.grid(row=2, column= 1)

 Start the main event loop
window.mainloop()
```

In this program, a frame is created using the `tk.Frame()` constructor. The `pack()` method is used to add the frame to the main window.

Two labels are created in the frame using the `tk.Label()` constructor. The `grid()` method is used to add the labels to the frame. The `row` and `column` parameters specify the position of the labels in the grid.

Two entry fields are created in the frame using the `tk.Entry()` constructor. The `grid()` method is used to add the entry fields to the frame. The `row` and `column` parameters specify the position of the entry fields in the grid. The `show` parameter is used to hide the password as it is being typed.
A button is created in the frame using the `tk.Button()` constructor. The `grid()` method is used to add the button to the frame. The `row` and `column` parameters specify the position of the button in the grid.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI

accordingly . When you run this program, you will see a window with a login form containing two labels, two entry fields, and a button. The labels are for the username and password fields, and the entry fields are where the user can enter their username and password. The button is used to submit the form.

You can use the `grid()` method to arrange widgets in a grid layout. The `row` and `column` parameters specify the position of the widget in the grid. You can also use the `sticky` parameter to specify how the widget should be aligned within its cell in the grid. For more information on grid layouts, you can refer to the Tkinter documentation:

https://www.tcl.tk/man/tcl8.6/TkCmd/grid.htm

97. Write a Python program to create a registration form in a GUI application.

```python
import tkinter as tk

 Create the main window
window =  tk.Tk()
window.title("Registration Form")

 Create a frame to hold the form
frame = tk.Frame(window)
frame .pack()

 Create a label for the name field
label_name = tk.Label(frame, text="Name:")
label_name.grid(row=0, column=0)

 Create an entry field for the name
entry_name = tk.Entry(frame)
entry _name.grid(row=0, column=1)

 Create a label for the email field
label_email = tk.Label(frame, text="Email:")
```

```
label_email.grid(row=1, column=0)

 Create an entry field for the email
entry_email = tk.Entry(frame)
entry_email.grid(row=1, column=1)

 Create a label for the password field
label_password = tk.Label(frame, text="Password:")
label_password.grid(row=2, column=0)

  Create an entry field for the password
entry_password = tk.Entry(frame, show="*")
entry_password.grid(row=2, column=1)

 Create a label for the confirm password field
label_confirm_password = tk.Label(frame, text="Confirm Password:")
label_confirm_password.grid(row=3, column=0)

 Create an entry field for the confirm password
entry_confirm_password = tk.Entry(frame, show="*")
entry_confirm_password.grid(row=3, column=1)

 Create a button to submit the form
button_submit = tk.Button(frame, text="Register")
button_submit.grid(row=4, column=1)

 Start the main event loop
window.mainloop()
```

In this program, a frame is created using the `tk.Frame()` constructor. The `pack()` method is used to add the frame to the main window.

Five labels are created in the frame using the `tk.Label()` constructor. The `grid()` method is used to add the labels to the frame. The `row` and `column` parameters specify the position of the labels in the grid.

Five entry fields are created in the frame using the `tk.Entry()` constructor. The `grid()` method is used to add the entry fields to the frame. The `row` and `column` parameters specify the position of the entry fields in the grid. The `show` parameter is used to hide the password and confirm password as they are being typed.

A button is created in the frame using the `tk.Button()` constructor. The `grid()` method is used to add the button to the frame. The `row` and `column` parameters specify the position of the button in the grid.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly. When you run this program, you will see a window with a registration form containing five labels, five entry fields, and a button. The labels are for the name, email, password, confirm password, and submit button. The entry fields are where the user can enter their name, email, password, and confirm password. The button is used to submit the form.

You can use the `grid()` method to arrange widgets in a grid layout. The `row` and `column` parameters specify the position of the widget in the grid. You can also use the `sticky` parameter to specify how the widget should be aligned within its cell in the grid.

For more information on grid layouts, you can refer to the Tkinter

documentation: https://www.tcl.tk/man/tcl8.6/TkCmd/grid.htm

98. Write a Python program to create a contact form in a GUI application.

```python
import tkinter as tk
 Create the main window
window =  tk.Tk()
window.title("Contact Form")
```

```
 Create a frame to hold the form
frame = tk.Frame(window)
frame .pack()

 Create a label for the name field
label_name = tk.Label(frame, text="Name:")
label_name.grid(row=0, column=0)

 Create an entry field for the name
entry_name = tk.Entry(frame)
entry _name.grid(row=0, column=1)

 Create a label for the email field
label_email = tk.Label(frame, text="Email:")
label_email.grid(row=1, column=0)

 Create an entry field for the email
entry_email = tk.Entry(frame)
entry_email.grid(row=1, column=1)

 Create a label for the message field
label_message = tk.Label(frame, text="Message:")
label_message.grid(row=2, column=0)

  Create a text area for the message
text_message = tk.Text(frame, height=10)
text_message.grid(row=2, column=1)

 Create a button to submit the form
button_submit = tk.Button(frame, text="Send")
button_submit.grid(row=3, column=1)

 Start the main event loop
window.mainloop()
```

In this program, a frame is created using the `tk.Frame()` constructor. The

`pack()` method is used to add the frame to the main window.

Three labels are created in the frame using the `tk.Label()` constructor. The `grid()` method is used to add the labels to the frame. The `row` and `column` parameters specify the position of the labels in the grid.

Three entry fields are created in the frame using the `tk.Entry()` constructor. The `grid()` method is used to add the entry fields to the frame. The `row` and `column` parameters specify the position of the entry fields in the grid.

A text area is created in the frame using the `tk.Text()` constructor. The `grid()` method is used to add the text area to the frame. The `row` and `column` parameters specify the position of the text area in the grid. The `height` parameter specifies the number of lines of text that the text area can hold.

A button is created in the frame using the `tk.Button()` constructor. The `grid()` method is used to add the button to the frame. The `row` and `column` parameters specify the position of the button in the grid.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly.

When you run this program, you will see a window with a contact form containing three labels, three entry fields, a text area, and a button. The labels are for the name, email, and message fields. The entry fields are where the user can enter their name and email address. The text area is where the user can enter their message. The button is used to submit the form. You can use the `grid()` method to arrange widgets in a grid layout. The `row` and `column` parameters specify the position of the widget in the grid. You can also use the `sticky` parameter to specify how the widget should be aligned within its cell in the grid.

For more information on grid layouts, you can refer to the Tkinter

documentation: https://www.tcl.tk/man/tcl8.6/TkCmd/grid.htm

99. Write a Python program to create a feedback form in a GUI application.

```python
import tkinter as tk

 #Create the main window
window =  tk.Tk()
window.title("Feedback Form")

 #Create a frame to hold the form
frame = tk.Frame(window)
frame .pack()

 #Create a label for the name field
label_name = tk.Label(frame, text="Name:")
label_name.grid(row=0, column=0)

 #Create an entry field for the name
entry_name = tk.Entry(frame)
entry _name.grid(row=0, column=1)

 #Create a label for the email field
label_email = tk.Label(frame, text="Email:")
label_email.grid(row=1, column=0)

 #Create an entry field for the email
entry_email = tk.Entry(frame)
entry_email.grid(row=1, column=1)

 #Create a label for the feedback field
label_feedback = tk.Label(frame, text="Feedback:")
label_feedback.grid(row=2, column=0)

  #Create a text area for the feedback
```

```
text_feedback = tk.Text(frame, height=10)
text_feedback.grid(row=2, column=1)

 #Create a button to submit the form
button_submit = tk.Button(frame, text="Send")
button_submit.grid(row=3, column=1)

# Start the main event loop
window.mainloop()
```

In this program, a frame is created using the `tk.Frame()` constructor. The `pack()` method is used to add the frame to the main window.

Three labels are created in the frame using the `tk.Label()` constructor. The `grid()` method is used to add the labels to the frame. The `row` and `column` parameters specify the position of the labels in the grid.

Three entry fields are created in the frame using the `tk.Entry()` constructor. The `grid()` method is used to add the entry fields to the frame. The `row` and `column` parameters specify the position of the entry fields in the grid.

A text area is created in the frame using the `tk.Text()` constructor. The `grid()` method is used to add the text area to the frame. The `row` and `column` parameters specify the position of the text area in the grid. The `height` parameter specifies the number of lines of text that the text area can hold.

A button is created in the frame using the `tk.Button()` constructor. The `grid()` method is used to add the button to the frame. The `row` and `column` parameters specify the position of the button in the grid.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly. When you run this program, you will see a window with a feedback form containing three labels, three entry fields, a text area, and a button. The labels are for the name, email, and feedback fields. The entry

fields are where the user can enter their name and email address. The text area is where the user can enter their feedback. The button is used to submit the form.

You can use the `grid()` method to arrange widgets in a grid layout. The `row` and `column` parameters specify the position of the widget in the grid. You can also use the `sticky` parameter to specify how the widget should be aligned within its cell in the grid.

For more information on grid layouts, you can refer to the Tkinter

documentation: https://www.tcl.tk/man/tcl8.6/TkCmd/grid.htm

100. Write a Python program to create a survey form in a GUI application.

```python
import tkinter as tk

 #Create the main window
window =  tk.Tk()
window.title("Survey Form")

 #Create a frame to hold the form
frame = tk.Frame(window)
frame .pack()

# Create a label for the name field
label_name = tk.Label(frame, text="Name:")
label_name.grid(row=0, column=0)

 #Create an entry field for the name
entry_name = tk.Entry(frame)
entry _name.grid(row=0, column=1)

 #Create a label for the email field
```

```
label_email = tk.Label(frame, text="Email:")
label_email.grid(row=1, column=0)

 #Create an entry field for the email
entry_email = tk.Entry(frame)
entry_email.grid(row=1, column=1)

 #Create a label for the survey question
label_question = tk.Label(frame, text="What is your favorite color?")
label_question.grid(row=2, column= 0)

 #Create a list of radio buttons for the survey question
radio_buttons = []
colors = ["Red", "Orange", "Yellow", "Green", "Blue", "Indigo", "Violet"]
for i, color in enumerate(colors):
    radio_button = tk.Radiobutton(frame, text=color, variable=tk.StringVar(),
value=color)
    radio_button.grid(row=i+3, column=0)
    radio_buttons.append(radio_button)

 #Create a button to submit the form
button_submit = tk.Button(frame, text="Submit")
button_submit.grid(row=len(colors)+3, column=1)

 #Start the main event loop
window.mainloop()
```

In this program, a frame is created using the `tk.Frame()` constructor. The `pack()` method is used to add the frame to the main window.

Three labels are created in the frame using the `tk.Label()` constructor. The `grid()` method is used to add the labels to the frame. The `row` and `column` parameters specify the position of the labels in the grid.

Three entry fields are created in the frame using the `tk.Entry()` constructor. The `grid()` method is used to add the entry fields to the frame. The `row` and

`column` parameters specify the position of the entry fields in the grid.

A list of radio buttons is created in the frame using the `tk.Radiobutton()` constructor. The `grid()` method is used to add the radio buttons to the frame. The `row` and `column` parameters specify the position of the radio buttons in the grid. The `variable` parameter specifies the variable that will be used to store the value of the selected radio button. The `value` parameter specifies the value that will be stored in the variable when the radio button is selected.

A button is created in the frame using the `tk.Button()` constructor. The `grid()` method is used to add the button to the frame. The `row` and `column` parameters specify the position of the button in the grid.

Finally, the `window.mainloop()` method is called to start the main event loop. This loop listens for events such as button clicks and updates the GUI accordingly. When you run this program, you will see a window with a survey form containing three labels, three entry fields, a list of radio buttons, and a button. The labels are for the name, email, and survey question fields. The entry fields are where the user can enter their name and email address. The radio buttons are for the user to select their favorite color. The button is used to submit the form. You can use the `grid()` method to arrange widgets in a grid layout. The `row` and `column` parameters specify the position of the widget in the grid. You can also use the `sticky` parameter to specify how the widget should be aligned within its cell in the grid.
For more information on grid layouts, you can refer to the Tkinter

documentation: https://www.tcl.tk/man/tcl8.6/TkCmd/grid.htm

The End

الختام

يمثل إكمال 100 مطالبة في "Python Prowess: Empower Your Journey with 100 Prompts" علامة بارزة في رحلتك لإتقان برمجة Python. من خلال هذه التحديات، قمت باستكشاف مجموعة متنوعة من المفاهيم، بدءًا من بناء الجملة الأساسي وحتى الموضوعات الأكثر تقدمًا، مما أدى إلى صقل مهاراتك وتعميق فهمك لـPython.

تم تصميم كل مطالبة بعناية لتجاوز حدودك، وتشجيعك على التفكير النقدي والإبداعي لحل المشكلات. على طول الطريق، من المحتمل أنك واجهت عقبات، وواجهت أخطاء، واحتفلت بالإنجازات. هذه التجارب كلها جزء من عملية التعلم، مما يساعدك على النمو كمبرمج.

ومن خلال المثابرة في تنفيذ جميع المطالبات المائة، فقد أظهرت التفاني والمرونة والشغف الحقيقي للتعلم. سواء كنت مبتدئًا تمامًا أو شخصًا لديه خبرة سابقة في البرمجة، فقد أدت هذه الرحلة بلا شك إلى توسيع كفاءتك في لغة Python وتزويدك بالأدوات اللازمة للتعامل مع مشاريع العالم الحقيقي بثقة.

عندما تفكر في إنجازاتك، تذكر أن تعلم بايثون لا يقتصر فقط على إتقان بناء الجملة أو حفظ الأوامر؛ يتعلق الأمر بتبني عقلية التحسين المستمر وحل المشكلات. ستكون المهارات التي طورتها من خلال هذه المطالبات بمثابة أساس متين للمساعي المستقبلية في تطوير البرمجيات وعلوم البيانات وتطوير الويب وما بعده.

لذلك، عندما تقلب الصفحة الأخيرة من"Python Prowess" ، افتخر بالمدى الذي وصلت إليه وتطلع إلى المغامرات المثيرة التي تنتظرك في رحلتك إلى Python. سواء كنت تقوم بإنشاء تطبيقات، أو تحليل البيانات، أو استكشاف مجالات جديدة، تذكر أن المعرفة والخبرة المكتسبة من هذه المطالبات المائة ستكون معك دائمًا، وتوجهك نحو مستويات أعلى من النجاح والابتكار. استمر في البرمجة، واستمر في الاستكشاف، ولا تتوقف أبدًا عن تحدي نفسك. عالم برمجة بايثون في انتظارك - انطلق وانتصر!

السيرة الذاتية للمؤلف

Falah G.Salih Resume

Education

1- B.Sc. In Physics' Science, University of Baghdad. *(1986-1987)*

2- Diploma in Ceramic Art in the Popular Arts Center/Baghdad (1995-1996).

3- Programmer from 1987 until now.

Computer Skills and programming languages:

1-Visual C++. And Visual Basic .Net

2- ASP Server Side Programming.

3-Java Script. for web pages.

4-Java for desktop.

5-MYSQL Server (Data Base systems).for IBM Co.

6- Developing Microsoft ASP.NET Web Application using Visual Studio.Net & ADO.NET Components for database systems.

7- Microsoft SQL Server (version 2000 & 2005) & Database Search Engines Systems.

8-PHP Server Side Programming (PHP Nuke and Forum for MS).

9- Static Pages Programming Languages (HTML & DHTML).

10- ASP.NET Server Side Programming with MS SQL Server.

11- Oracle SQL Database 10g

12– ArcView and Arc Map for GIS Application for spatial data analysis.

13-Microcontroller apps. (Arduino & Esp8266 MCU & Raspberry pi) 2014-

13- Android applications. (2011-)

14- Python for AI applications. (2017-)

15- Artificial intelligence in deep learning and computer vision applications.

16-flutter and dart for Android applications. (2020_)

17- AI Artificial Intelligence Model Developer (2018-) for AI Art Models Dev.

18- AI Artificial Intelligence Model Developer (2019-) for NLP Models.

19 -AI Artificial Intelligence Model Developer (2022-) for ChatBot Assistance Models.

Certificates:-

1- Microsoft Certified Professional (MCP). Mar 13, 2007.

2- Microsoft Certified Application Developer (MCAD). May 10 2007

https://tinyurl.com/2x3hfkwp

3- Microsoft Certified Solution Developer (MCSD). Aug. 12 2007.

4- Data Analysis with Python
5- Python 101 for Data Science
6- Deep Learning with TensorFlow

بأمكانك مشاهدة جميع الشهادات

مشاهدة جميع مشاريعي في المجالات التالية

- **Education field**
- **Health field**
- **Army field**
- **Industrial field**
- **General app.**
- **Agricultural app.**
- **Artificial Intelligence app.**

بأمكانك مشاهدة جميع مشاريعي

https://tinyurl.com/2p8cejwa

My website:

- **Blog: https://iraqprogrammer.wordpress.com**
- **Email: falahgs07@gmail.com**
- **Huggingface: https://huggingface.co/Falah**
- **AI4Art Models: https://civitai.com/user/falahgs/models**

- **YouTube: https://www.youtube.com/c/FalahgsGate**
- **Amazon: https://www.amazon.com/stores/author/B0BYHXLP7R/**
- **Github: https://github.com/falahgs**
- **PyPi: https://pypi.org/user/falahgs/**
- **Facebook: https://www.facebook.com/falahgs4ai**
- **Facebook: https://www.facebook.com/falahgs**
- **Telegram: https://t.me/falahgs_dl_cv**
- **LinkedIn: https://www.linkedin.com/in/falah-gatea-060a211a7/**
- **Twitter: https://twitter.com/FalahGatea**
- **NightCafe AI Art: https://creator.nightcafe.studio/u/FalahGS**
- **Artstation AI Art : https://www.artstation.com/falahgs**
- **Medium Posts: https://medium.com/@falahgs**
- **Instagram: https://www.instagram.com/falahgs4ai/**
- **https://www.instagram.com/falah.g.saleih/**

Package in PYPI python Package Index

1. https://pypi.org/project/multivision/
2. https://pypi.org/project/Kurdish2Image/
3. https://pypi.org/project/Image2Story/
4. https://pypi.org/project/ClipExtractor/
5. https://pypi.org/project/GPT3Prompts/
6. https://pypi.org/project/AudioText/
7. https://pypi.org/project/GoogleAudio/
8. https://pypi.org/project/CocoDataset/
9. https://pypi.org/project/Keras28Models/
10. https://pypi.org/project/StyleTransferArt/
11. https://pypi.org/project/ArabicOcr/
12. https://pypi.org/project/deep4dream/
13. https://pypi.org/project/pycovid19lstm/
14. https://pypi.org/project/Youtube2Images/
15. https://pypi.org/project/covid19forecast/
16. https://pypi.org/project/covid19-cases/

AI models for stable diffusions Text to Images

https://civitai.com/user/falahgs

1. https://civitai.com/models/135674/sdarchitecturalv1
2. https://civitai.com/models/46891/babylon
3. https://civitai.com/models/72965/zaha-hadid-style
4. https://civitai.com/models/158210/sdlogo-designer
5. https://civitai.com/models/137424/home-rooms-decoration
6. https://civitai.com/models/123529/fashion-model
7. https://civitai.com/models/222327/arabesque-visions-ai-powered-arabic-anime-art-generator
8. https://civitai.com/models/192584/sdmandala-style
9. https://civitai.com/models/140635/charcoal-style
10. https://civitai.com/models/93301/firuziat-female-style
11. https://civitai.com/models/131272/ayad-radhi
12. https://civitai.com/models/81514/baghdadi-heritage
13. https://civitai.com/models/91138/arabic-abaya
14. https://civitai.com/models/85086/arabic-style
15. https://civitai.com/models/62388/arabic-princess-style
16. https://civitai.com/models/70091/dishdasha-fashion
17. https://civitai.com/models/48636/kurdish-fashion
18. https://civitai.com/models/242475/sdxl-lora-photorealistic-female
19. https://civitai.com/models/239219/sdxl-lora-robotic
20. https://civitai.com/models/246826/sdxl-lora-zaha-hadid4architect
21. https://civitai.com/models/222327/arabesque-visions-ai-powered-arabic-anime-art-generator
22. https://civitai.com/models/249692/sdxl-lora-ants-style
23. https://civitai.com/models/254553/sdxl-lora-female-winter-magic
24. https://civitai.com/models/263530/sdxl-lora-3d-packages-design
25. https://civitai.com/models/273626/sdxl-lora-3d-cartoon

Datasets for AI Models Developing

1. https://huggingface.co/datasets/Falah/photography_prompts
2. https://huggingface.co/datasets/Falah/pixarstyle_prompts
3. https://huggingface.co/datasets/Falah/manga_art_style_prompts
4. https://huggingface.co/datasets/Falah/portrait_best_prompts
5. https://huggingface.co/datasets/Falah/portrait_prompts
6. https://huggingface.co/datasets/Falah/3d-school_prompts
7. https://huggingface.co/datasets/Falah/package_design_prompts
8. htps://huggingface.co/datasets/Falah/female_prompts
9. https://huggingface.co/datasets/Falah/human_generator_prompts
10. https://huggingface.co/datasets/Falah/framed_wall_art_prompts_SDXL
11. https://huggingface.co/datasets/Falah/modern_architectural_style_prompts_SDXL
12. https://huggingface.co/datasets/Falah/ancient_city_prompts_SDXL
13. https://huggingface.co/datasets/Falah/architecture_house_building_prompts_SDXL
14. https://huggingface.co/datasets/Falah/architecture_prompts_SDXL
15. https://huggingface.co/datasets/Falah/image_generation_prompts_SDXL
16. https://huggingface.co/datasets/Falah/cars_model_prompts_SDXL
17. https://huggingface.co/datasets/Falah/art_style_pareidolia_prompts_SDXL
18. https://huggingface.co/datasets/Falah/action_actor_prompts_SDXL
19. https://huggingface.co/datasets/Falah/sci_fi_fighter_plane_prompts_SDXL
20. https://huggingface.co/datasets/Falah/kids_fashion_prompts_SDXL
21. https://huggingface.co/datasets/Falah/men_fashion_prompts_SDXL
22. https://huggingface.co/datasets/Falah/interior_design_prompts_SDXL
23. https://huggingface.co/datasets/Falah/arabic_magical_fantasy_prompts_sdxl_refiner
24. https://huggingface.co/datasets/Falah/female_photo_prompts_sdxl_refiner
25. https://huggingface.co/datasets/Falah/movie_action_prompts_SDXL
26. https://huggingface.co/datasets/Falah/school_kids_prompts_SDXL
27. https://huggingface.co/datasets/Falah/cinematic_photo_prompts_SDXL
28. https://huggingface.co/datasets/Falah/fairy_girl_prompts_SDXL
29. https://huggingface.co/datasets/Falah/screamimg_portrait_prompts_sdxl
30. https://huggingface.co/datasets/Falah/arabic_islamic_fashion_prompts_SDXL
31. https://huggingface.co/datasets/Falah/hat_fashion_photography_prompts_SDXL
32. https://huggingface.co/datasets/Falah/screamimg_woman_portrait_prompts_sdxl
33. https://huggingface.co/datasets/Falah/fashion_photography_prompts_SDXL
34. https://huggingface.co/datasets/Falah/3M_baghdad_city_SDXL_refiner_prompts
35. https://huggingface.co/datasets/Falah/3M_future_baghdad_SDXL_refiner_prompts

36. https://huggingface.co/datasets/Falah/desert_arabic_fashion_SDXL_refiner_prompts
37. https://huggingface.co/datasets/Falah/desert_fashion_SDXL_refiner_prompts
38. https://huggingface.co/datasets/Falah/1M_luxury_yacht_SDXL_refiner_prompts
39. https://huggingface.co/datasets/Falah/2M_fantasy_SDXL_refiner_prompts
40. https://huggingface.co/datasets/Falah/2M_magic_nights_SDXL_refiner_prompts
41. https://huggingface.co/datasets/Falah/2M_fantastic_creatures_SDXL_refiner_prompts
42. https://huggingface.co/datasets/Falah/2M_creature_animales_SDXL_refiner_prompts
43. https://huggingface.co/datasets/Falah/2M_landscape_cities_SDXL_refiner_prompts
44. https://huggingface.co/datasets/Falah/2M_fashionable_girl_SDXL_refiner_prompts
45. https://huggingface.co/datasets/Falah/2M_arabic_architectural_futuristic_SDXL_refiner_prompts
46. https://huggingface.co/datasets/Falah/2M_arabic_female_SDXL_refiner_prompts
47. https://huggingface.co/datasets/Falah/2M_historical_flower_vase_SDXL_refiner_prompts
48. https://huggingface.co/datasets/Falah/2M_Ceramic_Vasa_SDXL_Refiner_Prompts
49. https://huggingface.co/datasets/Falah/1M_SDXL_Refiner_Prompts
50. https://huggingface.co/datasets/Falah/2000000_Style_art_prompts
51. https://huggingface.co/datasets/Falah/Ceramic_Style_art_SDXL
52. https://huggingface.co/datasets/Falah/mathematical_fashion_style_prompts
53. https://huggingface.co/datasets/Falah/Fibonacci_Golden_Ratio_Style_Prompts
54. https://huggingface.co/datasets/Falah/chairs_furniture
55. https://huggingface.co/datasets/Falah/stable_diffusion_prompts_dataset
56. https://huggingface.co/datasets/Falah/stable_diffusion_prompts
57. https://huggingface.co/datasets/Falah/Futuristic_prompts
58. https://huggingface.co/datasets/Falah/real_military_machinery_prompts
59. https://huggingface.co/datasets/Falah/3d-birds_animals_prompts
60. https://huggingface.co/datasets/Falah/military_machinery_prompts
61. https://huggingface.co/datasets/Falah/mobile_design_prompts
62. https://huggingface.co/datasets/Falah/presidents_prompts
63. https://huggingface.co/datasets/Falah/actor_action_prompts
64. https://huggingface.co/datasets/Falah/landscape_prompts
65. https://huggingface.co/datasets/Falah/robotic_prompts
66. https://huggingface.co/datasets/Falah/ghost_prompts
67. https://huggingface.co/datasets/Falah/skeletons_art_prompts
68. https://huggingface.co/datasets/Falah/ballet_dancing_style_art_prompts
69. https://huggingface.co/datasets/Falah/birds_animals_prompts
70. https://huggingface.co/datasets/Falah/islamic_prompts
71. https://huggingface.co/datasets/Falah/arabic_modern_prompts

72. https://huggingface.co/datasets/Falah/random_prompts
73. https://huggingface.co/datasets/Falah/war_prompts
74. https://huggingface.co/datasets/Falah/birds_animals_prompts
75. https://huggingface.co/datasets/Falah/photography_prompts
76. https://huggingface.co/datasets/Falah/ethereal_fantasy_prompts
77. https://huggingface.co/datasets/Falah/line_art_drawing_prompts
78. https://huggingface.co/datasets/Falah/marble_prompts
79. https://huggingface.co/datasets/Falah/sumerian_prompts
80. https://huggingface.co/datasets/Falah/expanded_artistic_prompts
81. https://huggingface.co/datasets/Falah/avatar_prompts
82. https://huggingface.co/datasets/Falah/fantasy_in_bottle
83. https://huggingface.co/datasets/Falah/Military_ships_prompts
84. https://huggingface.co/datasets/Falah/ads_corporate_prompts
85. https://huggingface.co/datasets/Falah/ads-fashion
86. https://huggingface.co/datasets/Falah/luxury_prompts
87. https://huggingface.co/datasets/Falah/food_photography
88. https://huggingface.co/datasets/Falah/ads-real_estate
89. https://huggingface.co/datasets/Falah/ads-retail
90. https://huggingface.co/datasets/Falah/ads-automotive
91. https://huggingface.co/datasets/Falah/magical_world_animals
92. https://huggingface.co/datasets/Falah/flower_arrangement
93. https://huggingface.co/datasets/Falah/animal_photorealistic
94. https://huggingface.co/datasets/Falah/arabic_enhanced_scenes
95. https://huggingface.co/datasets/Falah/new_photorealistic_prompts
96. https://huggingface.co/datasets/Falah/arabic_glamour_prompts
97. https://huggingface.co/datasets/Falah/instagram_model_ocean_grunge_prompts
98. https://huggingface.co/datasets/Falah/global_street_style_prompts
99. https://huggingface.co/datasets/Falah/global_elderly_woman_portrait_prompts
100. https://huggingface.co/datasets/Falah/micro_photography_subjects
101. https://huggingface.co/datasets/Falah/underwater_photography_subjects
102. https://huggingface.co/datasets/Falah/wildlife_photography_subjects
103. https://huggingface.co/datasets/Falah/cabo_da_roca_light_conditions
104. https://huggingface.co/datasets/Falah/photojournalism_fisherwoman
105. https://huggingface.co/datasets/Falah/samoan_fire_photography
106. https://huggingface.co/datasets/Falah/neo-pop_surrealism
107. https://huggingface.co/datasets/Falah/high_speed_photography
108. https://huggingface.co/datasets/Falah/retro_style_photography
109. https://huggingface.co/datasets/Falah/blonde_woman_photography_prompts

110.https://huggingface.co/datasets/Falah/product_photography_prompts
111.https://huggingface.co/datasets/Falah/catalogue_photography_prompts
112.https://huggingface.co/datasets/Falah/documentary_photography_prompts
113.https://huggingface.co/datasets/Falah/tilt_shift_photography_prompts
114.https://huggingface.co/datasets/Falah/luxurious_food_photography_prompts
115.https://huggingface.co/datasets/Falah/black_and_white_photography_prompts
116.https://huggingface.co/datasets/Falah/national_geographic_photography_prompts
117.https://huggingface.co/datasets/Falah/unsplash_photography_prompts
118.https://huggingface.co/datasets/Falah/vintage_photography_prompts
119.https://huggingface.co/datasets/Falah/time_lapse_photography_prompts
120.https://huggingface.co/datasets/Falah/fine_art_photography_prompts
121.https://huggingface.co/datasets/Falah/pinhole_photography_prompts
122.https://huggingface.co/datasets/Falah/female_runner_prompts
123.https://huggingface.co/datasets/Falah/close_up_shots_prompts
124.https://huggingface.co/datasets/Falah/profile_shots_prompts
125.https://huggingface.co/datasets/Falah/side_profile_portraits_prompts
126.https://huggingface.co/datasets/Falah/beach_back_angle_shots_prompts
127.https://huggingface.co/datasets/Falah/wide_angle_city_shots_prompts
128.https://huggingface.co/datasets/Falah/fish_eye_overlooking_industrial_site_prompts
129.https://huggingface.co/datasets/Falah/local_market_vendor_prompts
130.https://huggingface.co/datasets/Falah/toddler_smiling_low_angle_shots_prompts
131.https://huggingface.co/datasets/Falah/retro_style_high_angle_shots_prompts
132.https://huggingface.co/datasets/Falah/varied_portrait_prompts
133.https://huggingface.co/datasets/Falah/ancient_landscape_descriptions
134.https://huggingface.co/datasets/Falah/artist_rooms_descriptions
135.https://huggingface.co/datasets/Falah/vartist_workshop_descriptions
136.https://huggingface.co/datasets/Falah/cyborg_full_body_prompts
137.https://huggingface.co/datasets/Falah/anime_art_descriptions
138.https://huggingface.co/datasets/Falah/enhanced_anime_art_descriptions
139.https://huggingface.co/datasets/Falah/toy_figure_descriptions
140.https://huggingface.co/datasets/Falah/logo_prompts
141.https://huggingface.co/datasets/Falah/emotion_prompts
142.https://huggingface.co/datasets/Falah/fantasy_animal_prompts
143.https://huggingface.co/datasets/Falah/ali_prompts
144.https://huggingface.co/datasets/Falah/Islamic_forest_image_prompts
145.https://huggingface.co/datasets/Falah/cyberpunk_photo_prompts2
146.https://huggingface.co/datasets/Falah/mosque_forest_image_prompts
147.https://huggingface.co/datasets/Falah/photogram_prompts

148.https://huggingface.co/datasets/Falah/book_cover_prompts_with_text
149.https://huggingface.co/datasets/Falah/book_cover_prompts_with_sections
150.https://huggingface.co/datasets/Falah/programming_book_cover_prompts
151.https://huggingface.co/datasets/Falah/flowers_seed_prompts
152.https://huggingface.co/datasets/Falah/night_time_prompts
153.https://huggingface.co/datasets/Falah/kids_coloring_book_prompts
154.https://huggingface.co/datasets/Falah/coloring_book_animals
155.https://huggingface.co/datasets/Falah/invention_prompts
156.https://huggingface.co/datasets/Falah/islamic_invention_prompts
157.https://huggingface.co/datasets/Falah/character_prompts
158.https://huggingface.co/datasets/Falah/artwork_prompts
159.https://huggingface.co/datasets/Falah/character_prompts_arabic_best
160.https://huggingface.co/datasets/Falah/character_prompts_arabic
161.https://huggingface.co/datasets/Falah/artistic_prompts
162.https://huggingface.co/datasets/Falah/celebrity_prompts
163.https://huggingface.co/datasets/Falah/celebrity_art_prompts
164.https://huggingface.co/datasets/Falah/race_random_prompts
165.https://huggingface.co/datasets/Falah/architecture_prompts
166.https://huggingface.co/datasets/Falah/water_effects_prompts
167.https://huggingface.co/datasets/Falah/succulents_prompts
168.https://huggingface.co/datasets/Falah/wooden_objects_prompts
169.https://huggingface.co/datasets/Falah/movie_poster_prompts
170.https://huggingface.co/datasets/Falah/app_icon_prompts
171.https://huggingface.co/datasets/Falah/app_mockup_prompts
172.https://huggingface.co/datasets/Falah/ghibli_stills_prompts
173.https://huggingface.co/datasets/Falah/sneaker_concepts_prompts
174.https://huggingface.co/datasets/Falah/fashion_moodboards_prompts
175.https://huggingface.co/datasets/Falah/anime_arabic_style_prompts

Developer Books for AI