Python Basic Questions and Answers

Introduction

This document provides detailed answers to questions about Python programming, covering topics such as general Python features, data types, conditional statements, loops, modules, and libraries. Each answer is designed to clarify concepts and encourage deeper understanding through reflective questions.

1 Python Basics

1.1 What is Python?

Python is a high-level, interpreted, general-purpose programming language known for its readability and support for multiple paradigms (procedural, object-oriented, functional). Its clear syntax reduces the learning curve. Why do you think readability is important in programming?

1.2 What are the key features of Python?

Python is easy to read and write, interpreted (no compilation needed), dynamically typed (no need to declare variable types), supports multiple paradigms, has an extensive standard library, and a strong community. How might an extensive standard library benefit a developer?

1.3 What are the different data types in Python?

Common data types include integers (e.g., 5), floats (e.g., 3.14), strings (e.g., ”hello”), lists ([1, 2, 3]), tuples ((1, 2)), sets ({1, 2, 3}), and dictionaries ({”key”: ”value”}). Can you think of a scenario where you’d choose a tuple over a list?

1.4 What is a variable in Python?

A variable is a named location in memory that stores data, created by assigning a value to a name (e.g., x = 10). Think of it as a labeled box holding a value. Why might naming variables clearly improve code maintainability?

1.5 Benefits of using Python over other languages?

Python's clear syntax makes it easy to learn and write. It's versatile for web development, data analysis, machine learning, and automation. Its large standard library reduces external dependencies, and it's cross-platform, running unchanged on Windows, macOS, and Linux. Python is beginner-friendly and handles complex tasks like memory management. Which of these benefits would be most valuable for a data analysis project?

1.6 What are Python's built-in functions?

Python includes functions like print(), len(), type(), int(), str(), sum(), min(), and max() for common tasks. How might len() be useful when working with different data types?

1.7 How do you write comments in Python?

Single-line comments use # (e.g., # This is a comment). Multi-line comments use triple quotes (”””Comment here””” or ”’Comment here”’). Why are comments important for collaborative coding?

2 Lists

2.1 What is a list in Python?

A list is an ordered, mutable collection of items, which can have different types (e.g., [1, ”hello”, 3.14]). What does “ordered” mean in this context?

2.2 What does it mean for a list to be mutable?

Mutable means a list can be modified after creation—items can be added, removed, or changed. How does mutability affect how you use lists in a program?

2.3 What are the methods that a list contains?

List methods include:

- append(x): Adds item x to the end.
- remove(x): Removes the first occurrence of x.
- sort(): Sorts the list in place.
- clear(): Removes all items.
- count(x): Counts occurrences of x.
- reverse(): Reverses the list.
- copy(): Creates a shallow copy.
- pop([i]): Removes and returns the item at index i (or last item if no index).

Which method would you use to add multiple items to a list at once?

2.4 What are the functions that a list contains?

List-related functions include:

- len(list): Returns the number of items.
- max(list): Returns the largest item.
- min(list): Returns the smallest item.
- sum(list): Sums all items (for numbers).
- sorted(list, key=None, reverse=False): Returns a new sorted list.
- list(iterable): Converts an iterable to a list.

How does sorted() differ from sort()?

2.5 Can a list contain elements of different data types?

Yes, lists can hold mixed types (e.g., [1, "text", [2, 3]]). Why might this flexibility be useful?

2.6 What is a nested list?

A nested list is a list containing other lists (e.g., [[1, 2], [3, 4]]). How could nested lists represent a matrix?

2.7 What happens if you try to access an index that is out of range in a list?

It raises an IndexError. How could you prevent this error in your code?

2.8 How can you iterate over all the elements in a list?

Use a for loop: for element in my$_list$: .When might you use a loop versus indexing to access list elemen

2.9 How do you check if an item is in a list?

Use the in keyword: if item in my$_list$: .Why is this method efficient for checking membership?

2.10 What is the difference between insert() and append() methods in a list?

insert(i, x) adds x at index i, shifting other elements. append(x) adds x to the end. When would you need to use insert() instead of append()?

2.11 Can lists be used as keys in dictionaries?

No, lists are mutable and not hashable, so they cannot be dictionary keys. What data types can be used as dictionary keys?

2.12 How do you concatenate two lists in Python?

Use the + operator (e.g., list1 + list2). What's another way to combine lists?

2.13 What is the difference between append() and extend() methods in a list?

append(x) adds x as a single element to the end. extend(iterable) adds each item from iterable to the list. What happens if you append() a list versus extend() a list?

3 Strings

3.1 What is a string in Python?

A string is a sequence of characters enclosed in single (') or double (") quotes. Why might you choose one quote type over the other?

3.2 How do you access characters in a string?

Use indexing: s[0] gets the first character. What happens if you try to modify a character using indexing?

3.3 How do you find the length of a string?

Use len(s). How could you use this in a loop?

3.4 How do you concatenate strings in Python?

Use the + operator (e.g., "hello" + "world"). What's another way to combine strings?

3.5 How do you check if a substring exists in a string?

Use in: if "sub" in s:. How does this compare to searching in a list?

3.6 How do you convert a string to uppercase or lowercase?

Use s.upper() for uppercase, s.lower() for lowercase. Why might case conversion be useful in text processing?

3.7 How do you replace characters in a string?

Use s.replace(old, new). What happens if the substring to replace isn't found?

3.8 How do you join elements of a list into a single string?

Use "sep".join(list) (e.g., "-".join(["a", "b"]) yields a-b). What's the role of the separator in join()?

3.9 How do you check if a string starts or ends with a specific substring?

Use s.startswith(sub) or s.endswith(sub). When might these methods be useful in file processing?

3.10 How do you strip whitespace from the beginning and end of a string?

Use s.strip(). What's the difference between strip(), lstrip(), and rstrip()?

3.11 How do you count occurrences of a substring in a string?

Use s.count(sub). How could this help in text analysis?

3.12 How do you check if all characters in a string are alphanumeric?

Use s.isalnum(). What kind of input validation might use this?

3.13 How do you check if all characters in a string are digits?

Use s.isdigit(). How does isdigit() differ from isnumeric()?

3.14 How do you check if all characters in a string are alphabetic?

Use s.isalpha(). When might you need to check for alphabetic characters?

3.15 How do you check if a string is empty?

Use len(s) == 0 or not s. Why might you prefer one over the other?

3.16 How do you reverse a string?

Use slicing: s[::-1]. What does the -1 in the slice mean?

3.17 How do you convert a string to title case?

Use s.title(). How does title() handle multiple words?

3.18 How do you find the position of a substring within a string?

Use s.find(sub), which returns the index or -1 if not found. How does find() differ from index()?

3.19 How do you check if a string contains only whitespace?

Use s.isspace(). When might this be useful in data cleaning?

3.20 How do you check if a string is in lowercase or uppercase?

Use s.islower() or s.isupper(). What happens if a string contains mixed case?

3.21 How do you remove leading zeros from a string representing a number?

Use s.lstrip("0"). Why might this be important in numerical data processing?

4 Tuples

4.1 What is a tuple in Python?

A tuple is an ordered, immutable collection of elements in parentheses (e.g., (1, 2, "a")). Why is immutability valuable?

4.2 How do you create a tuple in Python?

Use parentheses: t = (1, 2, 3). How can you create a single-element tuple?

4.3 What is the main difference between a tuple and a list?

Tuples are immutable; lists are mutable. When would immutability be preferred?

4.4 Can a tuple contain different data types?

Yes, tuples can hold mixed types (e.g., (1, "text", 3.14)). How does this compare to lists?

4.5 How do you access elements in a tuple?

Use indexing: t[0]. What error occurs if the index is out of range?

4.6 Can you add or remove elements from a tuple?

No, tuples are immutable. How could you modify a tuple's contents indirectly?

4.7 What is the purpose of using a tuple over a list?

Tuples ensure data integrity for fixed collections. When might data integrity be critical?

4.8 How do you check the length of a tuple?

Use len(t). How does this compare to checking a list's length?

4.9 How do you convert a tuple to a list?

Use list(t). Why might you need to convert a tuple to a list?

4.10 How do you convert a list to a tuple?

Use tuple(list). When might you convert a list to a tuple?

4.11 What is tuple unpacking in Python?

Assign tuple elements to variables: a, b = (1, 2). How does unpacking simplify code?

4.12 How do you check if an element exists in a tuple?

Use in: if x in t:. How does this compare to list membership testing?

4.13 How do you iterate through elements of a tuple?

Use a for loop: for x in t:. When might you iterate over a tuple instead of a list?

4.14 Can a tuple be used as a dictionary key?

Yes, tuples are immutable and hashable. Why can't lists be dictionary keys?

4.15 Can you nest tuples inside other tuples?

Yes, e.g., (1, (2, 3)). How might nested tuples represent hierarchical data?

5 Dictionaries

5.1 What is a dictionary in Python?

A dictionary is an unordered collection of key-value pairs, where keys are unique (e.g., {"name": "Alice", "age": 25}). Why are key-value pairs useful?

5.2 How do you create a dictionary in Python?

Use curly braces: d = {"key": "value"}. What's another way to create a dictionary?

5.3 What are the characteristics of keys in a dictionary?

Keys must be unique and immutable (e.g., strings, numbers, tuples). Why must keys be immutable?

5.4 How do you access values in a dictionary?

Use keys: d["key"]. What happens if the key doesn't exist?

5.5 Can a dictionary contain duplicate keys?

No, assigning a new value to an existing key updates it. How does this ensure data consistency?

5.6 How do you add a new key-value pair to an existing dictionary?

Assign: d["new$_k$*ey*"] = *value*.What happens if the key already exists?

5.7 How do you remove a key-value pair from a dictionary?

Use del d[”key”] or d.pop(”key”). How do these methods differ?

5.8 What is the purpose of using a dictionary over a list?

Dictionaries provide fast key-based lookup. When might key-based access be faster than list indexing?

5.9 How do you get a list of all keys or values in a dictionary?

Use d.keys() for keys, d.values() for values. How can you convert these to lists?

5.10 How do you clear all elements from a dictionary?

Use d.clear(). What's the difference between clear() and creating a new dictionary?

5.11 Can a dictionary value be a list?

Yes, values can be any type, including lists. How might this be useful in data storage?

5.12 Can dictionaries be nested inside other dictionaries?

Yes, e.g., {”outer”: {”inner”: ”value”}}. How could nested dictionaries represent complex data?

5.13 What is the difference between dictionaries and sets?

Dictionaries store key-value pairs; sets store unique elements. When would you use a set instead of a dictionary?

5.14 How do you check if two dictionaries have the same key-value pairs?

Use d1 == d2. Does order matter in this comparison?

6 Sets

6.1 What is a set in Python?

A set is an unordered collection of unique, immutable elements in curly braces (e.g., {1, 2, 3}). Why are sets useful for unique data?

6.2 How do you create a set in Python?

Use curly braces: s = {1, 2, 3} or set([1, 2, 3]). How do you create an empty set?

6.3 Can a set contain duplicate elements?

No, duplicates are ignored. How does this help in data processing?

6.4 What are the characteristics of elements in a set?

Elements must be immutable (e.g., numbers, strings, tuples). Why is immutability required?

6.5 How do you convert a list to a set?

Use set(list). What happens to duplicates during conversion?

6.6 How do you access elements in a set?

Use loops or in, as sets are unordered and lack indexing. Why can't you use indexing?

6.7 How do you add elements to a set?

Use s.add(x). What happens if you add an existing element?

6.8 How do you remove elements from a set?

Use s.remove(x) (raises error if x is absent) or s.discard(x) (no error). When would you prefer discard()?

6.9 How do you check if an element exists in a set?

Use in: if x in s:. Why are sets efficient for membership testing?

6.10 How do you check if one set is a subset of another?

Use s1.issubset(s2). What's the relationship between subsets and supersets?

6.11 How do you check if two sets are equal?

Use s1 == s2. Why doesn't order matter in sets?

6.12 Can sets contain mutable elements like lists?

No, elements must be immutable for hashing. What types can be set elements?

6.13 Can you nest sets inside other sets?

No, sets are mutable and unhashable. How could you represent nested sets indirectly?

7 Conditional Statements

7.1 What are conditional statements in Python?

They execute code blocks based on conditions (True/False). Why are conditions essential in programming?

7.2 What are the types of conditional statements in Python?

if, elif, else. How do these work together?

7.3 How do you write an if statement in Python?

if condition: code. What makes a condition evaluate to True?

7.4 What happens if the condition in an if statement is False?

The code block is skipped. How can you handle the False case?

7.5 How do you use else with an if statement?

else: executes if the if condition is False. When is else optional?

7.6 Can you have multiple conditions in a sequence?

Yes, use elif for additional conditions. How does elif differ from multiple if statements?

7.7 How do you combine multiple conditions in an if statement?

Use and, or, not. How do these operators affect logic?

7.8 What is the purpose of the elif statement?

It checks additional conditions if previous ones are False. When might you need multiple elif statements?

7.9 How do you use nested if statements?

Place if inside another if. How can nesting affect code readability?

7.10 How do you check if a value is in a list before processing it?

Use if value in list:. Why is this check useful?

7.11 How do you compare two variables for equality in a conditional statement?

Use ==. How does == differ from is?

8 Loops

8.1 What are loops in Python?

Loops repeat code until a condition is met. Why are loops essential for automation?

8.2 What are the types of loops in Python?

for and while. When would you choose one over the other?

8.3 How do you write a for loop in Python?

for var in sequence:. What types of sequences can you iterate over?

8.4 How do you write a while loop in Python?

while condition:. What risks are associated with while loops?

8.5 What happens if the condition in a while loop is False from the beginning?

The loop body is skipped. How can you ensure a loop runs at least once?

8.6 What is a break statement?

break exits the loop early. When might you use break?

8.7 How do you terminate a loop early?

Use break. How does break affect loop control?

8.8 What is a continue statement?

continue skips the current iteration and proceeds to the next. When would continue be useful?

8.9 How do you skip the current iteration of a loop?

Use continue. How does continue differ from break?

8.10 What is a pass statement?

pass is a no-op placeholder. Why might you need a placeholder in a loop?

8.11 What is the range function?

range(start, stop, step) generates numbers. How does step affect the sequence?

8.12 How does range work in a for loop?

It generates numbers for iteration. What's the default start and step?

8.13 How do you iterate over a range of numbers using a for loop?

for i in range(5):. How can you customize the range?

8.14 How do you handle situations where no conditions are met in a loop?

Use else with a loop to run code if the loop completes without break. When might the else block be useful?

9 Modules and Packages

9.1 What is a module in Python?

A module is a file containing Python code (functions, classes, variables). How do modules promote code reuse?

9.2 How do you import a module in Python?

Use import module. What's the benefit of modular code?

9.3 What is the purpose of the import statement in Python?

It accesses code from other modules. How does this support collaboration?

9.4 How do you import a specific function from a module?

Use from module import function. Why might you import specific functions?

9.5 What is a package in Python?

A package is a directory of modules with an ___init___.py file. How do packages organize large projects?

9.6 What is the purpose of the ___init___.py file in a package?

It marks a directory as a package and can run initialization code. What might initialization code do?

9.7 What is the difference between a module and a package?

A module is a single file; a package is a collection of modules. When would you create a package?

9.8 What is the math module used for?

Provides mathematical functions (e.g., sin, log). What types of problems require the math module?

9.9 How do you list all available modules in Python?

Use help("modules"). How can you explore a module's contents?

9.10 What is the random module used for?

Generates random numbers (e.g., random.randint()). What applications use random numbers?

9.11 What is the os module used for?

Interacts with the operating system (e.g., file operations). What are some common os tasks?

9.12 What is the time module used for?

Handles time-related tasks (e.g., time.sleep()). When might you need to pause execution?

9.13 What is the datetime module used for?

Manipulates dates and times. How might this be used in data logging?

9.14 How do you document a module?

Use a docstring at the file's top. Why is documentation critical?

9.15 What is the re module used for?

Handles regular expressions for pattern matching. What's an example of a regex task?

9.16 What is a function in Python?

A reusable code block for a specific task. How do functions improve code organization?

9.17 How do you define a function in Python?

Use def name(args):. What makes a function reusable?

9.18 How do you call a function in Python?

Use name(args). What happens if you pass the wrong number of arguments?

9.19 What is the purpose of the return statement in a function?

Returns a value to the caller. What happens if there's no return?

10 *args and **kwargs

10.1 What are *args and **kwargs in Python?

*args collects positional arguments as a tuple; **kwargs collects keyword arguments as a dictionary. Why allow variable arguments?

10.2 What does *args do?

Collects positional arguments into a tuple. How does this help with flexibility?

10.3 What does **kwargs do?

Collects keyword arguments into a dictionary. When might keyword arguments be useful?

10.4 Can you use *args and **kwargs together in a function?

Yes, *args before **kwargs. Why is order important?

10.5 When should you use *args and **kwargs?

Use *args for unknown numbers of positional arguments; **kwargs for keyword arguments. What's an example use case?

10.6 What is the difference between *args and **kwargs?

*args is for positional arguments (tuple); **kwargs is for keyword arguments (dictionary). How do these differ in function calls?

10.7 What is the map() function in Python?

map(func, iterable) applies func to each item in iterable. Why use map() over a loop?

10.8 How do you use the map() function?

Pass a function and iterable: map(str, [1, 2, 3]). What types of functions can map() use?

10.9 Can you use map() with a lambda function?

Yes, e.g., map(lambda x: x*2, [1, 2, 3]). Why are lambda functions convenient here?

10.10 What does map() return?

A map object, convertible to a list with list(). Why is it a map object instead of a list?

10.11 Why use map() instead of a loop?

map() is concise and sometimes faster. When might a loop be clearer?

10.12 What is a lambda function in Python?

A small, anonymous function: lambda args: expression. Why use anonymous functions?

10.13 How do you create a lambda function?

lambda x: x*2. What's the limitation of lambda functions?

10.14 Can lambda functions take multiple arguments?

Yes, e.g., lambda x, y: x + y. How does this affect their use?

10.15 What is an exception in Python?

An error during execution. What causes exceptions?

10.16 What is exception handling?

Managing errors with try/except to prevent crashes. Why is graceful error handling important?

10.17 Why is exception handling important?

Prevents crashes and allows error recovery. What's an example of error recovery?

10.18 What is the try block?

Contains code that might raise an exception. Why isolate risky code?

10.19 What is the except block?

Handles specific exceptions. How does this improve robustness?

10.20 What is the finally block?

Runs regardless of exceptions. When might you need finally?

10.21 What is the else block?

Runs if no exception occurs. How does else clarify code?

10.22 Can you have multiple except blocks?

Yes, to handle different exceptions. Why handle specific exceptions?

10.23 What is a specific exception?

A named error type (e.g., ZeroDivisionError). Why catch specific exceptions?

10.24 What happens if no exception is raised in the try block?

The else block runs (if present). How does this differ from finally?

10.25 Can you use finally without except?

Yes, with try. When might this be useful?

10.26 Can you use try without except?

No, try requires except or finally. Why is this enforced?

10.27 Is it a good practice to catch all exceptions?

No, catching specific exceptions avoids hiding bugs. What risks come with catching all exceptions?

11 NumPy

11.1 What is NumPy?

A library for array operations with mathematical functions. Why are arrays useful in scientific computing?

11.2 Why is NumPy useful?

It's efficient for large datasets and provides mathematical functions. How does efficiency impact performance?

11.3 What is an array in NumPy?

A multidimensional ndarray for storing data. How does this differ from a list?

11.4 What is the difference between a Python list and a NumPy array?

Arrays are faster, support more operations, and are memory-efficient. Why is memory efficiency important?

11.5 What is the shape of an array?

The number of elements in each dimension (e.g., (2, 3)). How does shape affect array operations?

11.6 How do you create an array of zeros?

Use np.zeros(shape). When might you need an array of zeros?

11.7 How do you create an array of ones?

Use np.ones(shape). What's a use case for this?

11.8 How do you create an array with a range of numbers?

Use np.arange(start, stop, step). How does this compare to range()?

11.9 How do you create an array with random numbers?

Use np.random.random(shape). What's a random array useful for?

11.10 How do you add two arrays together?

Use np.add(arr1, arr2) or arr1 + arr2. Why is element-wise addition useful?

11.11 How do you find the maximum value in an array?

Use np.max(arr). How might this help in data analysis?

11.12 How do you find the minimum value in an array?

Use np.min(arr). When would you need the minimum?

11.13 How do you find the sum of all elements in an array?

Use np.sum(arr). What's a practical use for this?

11.14 How do you find the average of all elements in an array?

Use np.mean(arr). How does this relate to statistical analysis?

11.15 How do you find the standard deviation of elements in an array?

Use np.std(arr). Why measure standard deviation?

11.16 How do you slice a NumPy array?

Use arr[start:stop:step]. How does slicing help in data processing?

11.17 How do you concatenate two NumPy arrays?

Use np.concatenate((arr1, arr2)). When might you combine arrays?

11.18 How do you save a NumPy array to a file?

Use np.save("file", arr). Why save arrays to files?

11.19 How do you load a NumPy array from a file?

Use np.load("file.npy"). How does this support data persistence?

11.20 What is a multidimensional array in NumPy?

An array with multiple dimensions (e.g., a matrix). How are multidimensional arrays used in machine learning?

12 Pandas

12.1 What is Pandas in Python?

A library for data manipulation using Series and DataFrames. Why are tables useful for data analysis?

12.2 Why is Pandas preferred for data analysis in Python?

It offers easy data manipulation, cleaning, alignment, and grouping. How does handling missing data improve analysis?

12.3 What is a Series in Pandas?

A labeled one-dimensional array. How does a Series compare to a list?

12.4 What is a DataFrame in Pandas?

A table with rows and columns. How does a DataFrame resemble a spreadsheet?

12.5 How do you get basic statistical details of a DataFrame?

Use df.describe(). What statistics are included?

12.6 How do you read a CSV file using Pandas?

Use pd.read$_csv$("$file.csv$").Why is CSV a common format?

12.7 How do you display the first few rows of a DataFrame?

Use df.head(). Why check the first few rows?

12.8 How do you get the number of rows and columns in a DataFrame?

Use df.shape. How does this help in data exploration?

12.9 What is the difference between loc and iloc in Pandas?

loc uses labels; iloc uses integer positions. When would you use loc over iloc?

12.10 What is the purpose of the groupby function in Pandas?

Splits data, applies a function, and combines results. What's a common groupby operation?

12.11 How does Pandas handle missing data?

Use isnull(), dropna(), or fillna(). Why is handling missing data critical?

12.12 What is a pivot table in Pandas?

A tool to summarize data using pd.pivot$_t able()$.How does this help in reporting?

13 Matplotlib

13.1 What is Matplotlib?

A library for creating visualizations. Why visualize data?

13.2 What is the purpose of the pyplot module in Matplotlib?

Provides a simple interface for plotting. How does this simplify plotting?

13.3 How do you create a simple line plot?

Use plt.plot(x, y). What data is suitable for line plots?

13.4 What does the show function do in Matplotlib?

Displays the plot: plt.show(). Why is this needed?

13.5 How do you add a title to a plot?

Use plt.title("Title"). Why is a title important?

13.6 How do you label the x-axis and y-axis?

Use plt.xlabel("X") and plt.ylabel("Y"). How do labels improve clarity?

13.7 How do you create a scatter plot?

Use plt.scatter(x, y). When are scatter plots useful?

13.8 How do you add a legend to a plot?

Use plt.legend(). How does a legend aid interpretation?

13.9 What is a subplot?

A plot within a figure for multiple plots. Why use subplots?

13.10 How do you save a plot as an image file?

Use plt.savefig("file.png"). Why save plots?

13.11 How do you fill an area between two curves in a plot?

Use plt.fill$_b etween(x, y1, y2)$.What's a use case for this?

13.12 How do you create a horizontal bar chart?

Use plt.barh(y, width). When are horizontal bars preferred?

14 Seaborn

14.1 What is Seaborn?

A visualization library based on Matplotlib for statistical plots. How does Seaborn enhance Matplotlib?

14.2 What is the primary purpose of Seaborn?

Creates visually appealing statistical plots. Why focus on statistical graphics?

14.3 What are some common plot types you can create with Seaborn?

Scatter, line, bar, histogram, box, heatmap. Which plot suits categorical data?

14.4 How do you create a simple scatter plot with Seaborn?

Use sns.scatterplot(x="x", y="y", data=df). Why use a DataFrame here?

14.5 How do you create a simple line plot with Seaborn?

Use sns.lineplot(x="x", y="y", data=df). When are line plots useful?

14.6 How do you create a simple bar plot with Seaborn?

Use sns.barplot(x="x", y="y", data=df). How do bar plots show comparisons?

14.7 How do you create a histogram with Seaborn?

Use sns.histplot(data=df, x="x"). Why visualize distributions?

14.8 How do you create a box plot with Seaborn?

Use sns.boxplot(x="x", y="y", data=df). What does a box plot show?

14.9 What is the purpose of the pairplot function in Seaborn?

Creates scatter plot matrices for pairwise relationships. How does this help in data exploration?

14.10 How do you customize the color palette in Seaborn?

Use sns.set$_palette$("$palette$").Why customize colors?

14.11 What is the purpose of the distplot function in Seaborn?

Deprecated; use histplot or displot for distributions. Why was distplot replaced?

14.12 How do you save a Seaborn plot as an image file?

Use plt.savefig("file.png"). How does this integrate with Matplotlib?

14.13 What is the purpose of the facetgrid function in Seaborn?

Creates multiple plots by categorical variables. How does this aid analysis?

14.14 What is the purpose of the lmplot function in Seaborn?

Creates scatter plots with regression lines. When is regression visualization useful?

14.15 What is the purpose of the stripplot function in Seaborn?

Creates scatter plots for categorical variables. How does this differ from scatterplot?

14.16 What is the purpose of the heatmap function in Seaborn?

Visualizes data with colors in a matrix. When are heatmaps useful?

15 Statistics Module

15.1 What is the statistics module in Python?

Provides functions for statistical calculations. Why use a dedicated statistics module?

15.2 What function is used to calculate the mean of a list of numbers?

statistics.mean(data). What does the mean represent?

15.3 What function is used to calculate the median of a list of numbers?

statistics.median(data). How does the median differ from the mean?

15.4 What function is used to calculate the mode of a list of numbers?

statistics.mode(data). When is the mode useful?

15.5 What function would you use to find the central tendency of categorical data?

statistics.mode(data). Why is mode suitable for categorical data?

15.6 What is the pstdev function used for?

statistics.pstdev(data) calculates population standard deviation. What's the population in this context?

15.7 What is the stdev function used for?

statistics.stdev(data) calculates sample standard deviation. How does this differ from pstdev?

15.8 What is the pvariance function used for?

statistics.pvariance(data) calculates population variance. What does variance measure?

15.9 What is the variance function used for?

statistics.variance(data) calculates sample variance. Why use sample variance?

15.10 What is the median$_{l}ow function used for$?

statistics.median$_{l}ow(data) returns the lower median.$When is the lower median relevant?

15.11 What is the median$_{h}igh function used for$?

statistics.median$_{h}igh(data) returns the higher median.$How does this differ from median?

15.12 What is the median$_{g}rouped function used for$?

statistics.median$_{g}rouped(data) calculates the median of grouped data.$What's grouped data?

15.13 What is the quantiles function used for?

statistics.quantiles(data) divides data into equal-probability intervals. How are quantiles used in statistics?

15.14 What function would you use to measure the spread of a dataset?

statistics.variance() or stdev(). Why measure spread?

15.15 How do you handle an empty list with the mean function?

Raises StatisticsError. How can you prevent this error?

15.16 What is the mean function's return type?

Float. Why is a float appropriate for the mean?