CHAPTER 1: -

- Operators: *, -, /, + Values: 'hello', -88.8, 5
- Variable: spam String: 'spam'
- Three data types: Integer, Float, String
- An expression is made up of operators and values that can be evaluated to produce a result. All expressions return a value.
- An expression produces a value when evaluated, whereas a statement performs an action. Assignment statements like "spam = 10" are statements.
- After running the code: bacon = 20 bacon + 1 The variable "bacon" still contains 20.
- 'spam' + 'spamspam' evaluates to 'spamspamspam' 'spam' * 3 evaluates to 'spamspamspam'
- Variable names must start with a letter. 100 starts with a digit.
- int(), float(), str()
- The error is due to trying to concatenate a string ('I have eaten ') with an integer (99). You can fix it by converting 99 to a string: 'I have eaten ' + str(99) + ' burritos.'

CHAPTER 2 : -

1) Two Boolean values: True and False.
2) Three Boolean operators: and, or, not.
5) Six comparison operators: == (equal), != (not equal), > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to).
6) Equal to operator (==) compares values for equality, while the assignment operator (=) assigns a value to a variable.
7)A condition is a Boolean expression used to make decisions in programming. It's used in control structures like if statements and loops to determine the flow of execution based on whether the condition is true or false.
10)Press Ctrl+C.
11) `break` is used to exit a loop completely, while **`continue`** is used to skip the rest of the current iteration and move to the next iteration.
12) `range(10)` generates numbers from 0 to 9. All three forms are equivalent.

14) If you had a function named `bacon()` inside a module named `spam`, you would call it after importing `spam` using `spam.bacon()`.

CHAPTER 3: -

1. Functions are advantageous in programs because they allow for code reusability, modularity, and organization. They help in avoiding redundancy and making complex programs more manageable.
2. The code in a function executes when the function is called, not when it's defined.
3. The **`def`** statement creates a function in Python.
4. A function is a block of code that performs a specific task, while a function call is the act of using that function to execute its code.
5. There is one global scope in a Python program. Local scopes are created when functions are called, so there can be multiple local scopes, each associated with a function call.
6. Variables in a local scope are destroyed when the function call returns, and the local scope is destroyed.
7. A return value is the value that a function call evaluates to. Yes, a return value can be part of an expression.
8. If a function does not have a return statement, its return value is **`None`**.
9. You can use the **`global`** keyword inside the function to indicate that a variable should refer to the global variable instead of creating a new local variable.
10. The data type of **`None`** is **`NoneType`**.
11. The statement **`import areallyourpetsnamederic`** doesn't have a predefined effect in Python. It would raise an ImportError unless you have a module named **`areallyourpetsnamederic`**.
12. If you had a function named **`bacon()`** in a module named **`spam`**, you would call it after importing **`spam`** using **`spam.bacon()`**.
13. You can prevent a program from crashing by handling errors using try-except blocks. This way, if an error occurs, the program can gracefully handle the error and continue executing.
14. In a **`try`** clause, you place the code that you suspect might raise an exception. In an **`except`** clause, you handle the exception by providing the code to execute if the associated error occurs.

CHAPTER 4: -

- **`[]`** represents an empty list in Python.
- To assign 'hello' as the third value in the list stored in the variable **`spam`**:

- spam[2] = 'hello'
- **spam[int(int('3' * 2) // 11)]** evaluates to **'d'**.
- **spam[-1]** evaluates to **'d'**.
- **spam[:2]** evaluates to **['a', 'b']**.
- **bacon.index('cat')** evaluates to **1**.
- After **bacon.append(99)**, the list value in **bacon** becomes **[3.14, 'cat', 11, 'cat', True, 99]**.
- After **bacon.remove('cat')**, the list value in **bacon** becomes **[3.14, 11, 'cat', True, 99]**.
- List concatenation operator: **+** List replication operator: *****
- The **append()** method adds a value to the end of the list. The **insert()** method inserts a value at a specific index in the list.
- Two ways to remove values from a list are using the **remove()** method and the **del** statement.
- Both list values and string values can be indexed and sliced. They can also be looped over using loops like **for**.
- Lists are mutable (can be modified), while tuples are immutable (cannot be modified).
- Tuple containing the integer value 42: **(42,)**
- To get the tuple form of a list value: **tuple(some_list)** To get the list form of a tuple value: **list(some_tuple)**
- Variables that "contain" list values store references (pointers) to the memory location where the list data is stored.
- **copy.copy()** performs a shallow copy of a list, meaning it creates a new list, but the elements within it may still refer to the same objects as the original list. **copy.deepcopy()** performs a deep copy, creating entirely new objects for the copied list, including all nested objects.

CHAPTER 5: -

- An empty dictionary is represented by **{}**.

- A dictionary value with a key 'foo' and a value 42 looks like **`{'foo': 42}`**.
- The main difference between a dictionary and a list is that a dictionary stores values as key-value pairs, while a list stores values in an ordered sequence.
- If you try to access **`spam['foo']`** when **`spam`** is **`{'bar': 100}`**, a **`KeyError`** will be raised since the key **`'foo'`** is not present in the dictionary.
- If a dictionary is stored in **`spam`**, **`'cat' in spam`** checks if the key **`'cat'`** exists in the dictionary, while **`'cat' in spam.keys()`** does the same thing, checking for the presence of the key **`'cat'`**.
- If a dictionary is stored in **`spam`**, **`'cat' in spam`** checks if there's a key **`'cat'`** in the dictionary, while **`'cat' in spam.values()`** checks if there's a value **`'cat'`** in any of the dictionary's values.

CHAPTER 6: -

- Escape characters are special characters used in strings to represent characters that are difficult or impossible to type directly, such as newline or tab characters.
- **`\n`** represents a newline character, and **`\t`** represents a tab character.
- You can use a double backslash **`\\`** to represent a single backslash character in a string.
- The string is valid because the single quote character within the double-quoted string doesn't conflict with the string's outer double quotes. Alternatively, you could use single quotes around the entire string: 'Howl's Moving Castle'.
- You can use triple-quoted strings (**`'''`** or **`"""`**) to write strings with newlines in them.
- Expression evaluations:
- 'Hello, world!'[1] -> 'e'
    - 'Hello, world!'[0:5] -> 'Hello'
- 'Hello, world!'[:5] -> 'Hello'
- 'Hello, world!'[3:] -> 'lo, world!'
- Expression evaluations:
- 'Hello'.upper() -> 'HELLO'
- 'Hello'.upper().isupper() -> True
- 'Hello'.upper().lower() -> 'hello'
- Expression evaluations:
- 'Remember, remember, the fifth of November.'.split() -> ['Remember,', 'remember,', 'the', 'fifth', 'of', 'November.']
- '-'.join('There can be only one.'.split()) -> 'There-can-be-only-one.'
- String methods for justification:
- Right-justify: **`rjust()`**
- Left-justify: **`ljust()`**
- Center: **`center()`**
- You can use the **`strip()`** method to remove whitespace characters from the beginning and end of a string. To remove only leading or trailing whitespace, you can use **`lstrip()`** and **`rstrip()`** respectively.

CHAPTER 7: -

- The **re.compile()** function creates Regex objects in Python's **re** module.
- Raw strings (preceded by **r**) are often used with Regex objects to avoid having to escape backslashes twice. They treat backslashes as literal characters instead of escape characters.
- The **search()** method returns a Match object if the pattern is found in the searched string, otherwise, it returns **None**.
- You can use the **group()** method on a Match object to get the actual strings that match the pattern. **group(0)** is the entire matched text, and **group(1)**, **group(2)**, etc. refer to the capture groups.
- In the regex **r'(\d\d\d)-(\d\d\d-\d\d\d\d)'**:
- Group 0 covers the entire matched text.
- Group 1 covers the first three digits.
- Group 2 covers the second set of three digits and the final four digits.
- To match parentheses and period characters literally, you need to escape them with a backslash: **\(** for **(** and **\.** for **.**.
- The **findall()** method returns a list of strings when the regex has no groups, and it returns a list of tuples of strings when the regex has groups.
- The **|** character in regular expressions is used to signify an alternation, allowing you to match one expression or another.
- The **?** character signifies:
- Making the preceding element in the regex optional.
- Non-greedy matching when used after *****, **+**, **?**, or **{}** quantifiers.
- In regular expressions:
- **+** means "one or more occurrences."
- ***** means "zero or more occurrences."
- **{3}** specifies exactly 3 occurrences, while **{3,5}** specifies a range of 3 to 5 occurrences.
- Shorthand character classes in regular expressions:
- **\d** matches any digit.
- **\w** matches any word character (alphanumeric and underscore).

CHAPTER 8: -

- **\s** matches any whitespace character.
- Shorthand character classes in regular expressions:
- **\D** matches any non-digit character.
- **\W** matches any non-word character.
- **\S** matches any non-whitespace character.

- No, **PyInputPlus** does not come with the Python Standard Library. It is a third-party module that needs to be installed separately.
- **PyInputPlus** is often imported with **import pyinputplus as pyip** to provide a shorter alias that makes it easier to call its functions.
- **inputInt()** is used for input validation, specifically for integers, and raises an exception if the entered value is not an integer. **inputFloat()** does the same for floating-point numbers.
- You can ensure that the user enters a whole number between 0 and 99 using the **inputInt()** function with the **min** and **max** keyword arguments: **pyip.inputInt(prompt='Enter a number: ', min=0, max=99)**
- The **allowRegexes** and **blockRegexes** keyword arguments are lists of regular expression strings. **allowRegexes** specifies patterns that are allowed, and **blockRegexes** specifies patterns that are not allowed.
- **inputStr(limit=3)** will raise a **TimeoutException** if blank input is entered three times consecutively.
- **inputStr(limit=3, default='hello')** will return the default value **'hello'** if blank input is entered three times consecutively.