**Basic Concepts**

* interpreter, compiler, Python language properties, keywords, indents

Interpreter vs Compiler: Python is an interpreted language, which means Python code is executed by an interpreter at runtime. This is different from a compiled language where the code is first converted to machine language before it is executed by the computer's CPU.

Python Language Properties: Python is known for being readable and simple to learn. Its syntax is clean and emphasizes readability.

Keywords: These are reserved words in Python that have special meaning. For example, if, for, while, break, continue, etc.

Indents: Indentation is significant in Python and is used to define the scope of loops, functions, and classes.

* literals: logical, integer, float number, scientific display of numbers (eg 3.4 \* 10 ^ -3 and not 0.0034), text  
  Logical: True and False are boolean literals in Python.

Integer: Whole numbers such as 1, 2, 42.

Float number: Numbers with a decimal point, for example, 3.14.

Scientific Display of Numbers: This is used to represent very large or very small numbers. For example, 3.4e-3 is equivalent to 3.4 \* 10^-3.  
Text: String literals, enclosed in quotes: "Hello, World!"

* comment, print, input

Comment: Used to include descriptive text in the code that is not executed. In Python, you use # for a single-line comment.

Print: A function to display the output to the console: print("Hello, World!").

Input: A function to get input from the user: age = input("Enter your age: ").

* numeral systems

Python can represent numbers in different bases: binary (0b or 0B), octal (0o or 0O), hexadecimal (0x or 0X), and decimal (no prefix).

* operators with numbers (\*\*, \*, /,%, //, +, -)  
  \*\*: Exponentiation (e.g., 2 \*\* 3 is 8)

\*: Multiplication (e.g., 3 \* 4 is 12)

/: Division (e.g., 8 / 2 is 4.0)

%: Modulo (e.g., 5 % 2 is 1)

//: Floor division (e.g., 5 // 2 is 2)

+: Addition (e.g., 2 + 3 is 5)

-: Subtraction (e.g., 5 - 2 is 3)

* operators with strings (\*, +)  
  \*: Repetition (e.g., "hello" \* 2 is "hellohello")

+: Concatenation (e.g., "hello" + " " + "world" is "hello world")

* assignment operators (- =, \* =, + =, / =,% =)  
  =: Simple assignment (e.g., x = 5)

-=: Subtraction assignment (e.g., x -= 2 is equivalent to x = x - 2)

\*=: Multiplication assignment (e.g., x \*= 3 is equivalent to x = x \* 3)

+=: Addition assignment (e.g., x += 2 is equivalent to x = x + 2)

/=: Division assignment (e.g., x /= 2 is equivalent to x = x / 2)

%=: Modulo assignment (e.g., x %= 3 is equivalent to x = x % 3)

**Data types, evaluation**

* bitwise operators (~, &, ^, |, <<, >>)  
  NOT: ~

AND: &

XOR: ^

OR: |

Left Shift: <<

Right Shift: 32 >> 4

* logic operators (not, and, or)
* logic expressions (expressions that evaluate to either True or False using logic operators)
* relational operators (==, !=, >, >=, <, <=)  
  Equal to: ==  
  Not equal to: !=  
  Greater than: >  
  Greater than or equal to: >=  
  Less than: <  
  Less than or equal to: <=
* accuracy of floating point numbers (floating-point numbers come with some precision issues. This is due to the way these numbers are stored internally, which can lead to unexpected results (e.g., 0.1 + 0.2 may not exactly equal 0.3).)
* input(), print(), int(), float(), str(), len()

input(): For getting user input as a string.

print(): For displaying output.

int(): For converting to an integer.

float(): For converting to a float.

str(): For converting to a string.

len(): For getting the length of a collection or string.

* print formatting, end=, sep=

you can format strings with placeholders: print("Hello, %s!" % name).

End and Sep in Print: end parameter is used to specify what to print at the end of the line. By default, it's \n, a newline character. sep parameter is used to specify the separator between the items if there are multiple items in the print statement.

* types (several built-in data types such as int, float, str, list, tuple, dict, set, etc.)
* basic calculations   
  basic calculations are performed using arithmetic operators. These include addition (+), subtraction (-), multiplication (\*), division (/), modulus (% which gives the remainder of a division), exponentiation (\*\*), and floor division (// which divides and rounds down to the nearest integer).
* use of strings  
  -Strings in Python are sequences of characters enclosed in quotes (either single ' or double "). Python provides various operations that can be performed on strings such as indexing, slicing, concatenation (+ operator to join strings), and repetition (\* operator to repeat strings).

-Strings are immutable, which means once a string is created, the characters within it cannot be changed. However, you can create new strings based on operations performed on existing ones.

-Python also provides numerous methods for string manipulation, such as upper() for converting to uppercase, lower() for lowercase, strip() for removing whitespace, and many more.

-String formatting is a powerful feature in Python that allows you to embed variables within a string. There are several ways to format strings, including the older % operator, the str.format() method, and f-strings (formatted string literals) which were introduced in Python 3.6.

**Control flow**

**· statements (if, if-else, if-elif, if-elif-else) :**

* if Statement: Used for conditional execution. It checks a condition, and if it's true, it executes a block of code.
* if-else Statement: Extends the `if` statement to include an alternative block of code to execute if the condition is false.
* if-elif-else Statement: Allows for checking multiple conditions sequentially.

· **Nested Statements**:   
Statements (including `if`, `else`, `elif`, `while`, `for`, etc.) can be nested inside one another.

· **pass Expression**:   
The `pass` statement is a no-operation statement. It is often used as a placeholder where syntactically some code is required but no action is desired.

· **loops**:

* while Loop: Executes a block of code as long as the specified condition is true.
* for Loop: Iterates over a sequence (such as a list, tuple, or string) or other iterable objects.
* range Function: Generates a sequence of numbers that can be used in a `for` loop.
* in Keyword: Used in `for` loops to iterate over elements of a sequence.

· **Iterations in Sequences**: Loops can iterate through elements in sequences like strings or lists.

· **Extended Loops**:

- while-else: The `else` block is executed if the `while` loop condition becomes false.

- for-else: The `else` block is executed when the `for` loop completes its iterations.

· **Nested Loops, Statements**: Loops and statements can be nested inside each other to create more complex control flow structures.

· **Controlling Loops**:

* break: Terminates the loop prematurely.
* continue: Skips the rest of the code inside the loop for the current iteration and moves to the next iteration.

**Data - List, Tuple, Dictionary**

**Lists**:

* **simple Lists, Indexing, Slicing, len():**

```

my\_list = [1, 2, 3, 4, 5]

print(my\_list[0]) # Accessing by index

print(my\_list[1:3]) # Slicing

print(len(my\_list)) # Length of the list

```

* **append(), insert(), index(), sorted(), del, for loop, in, not in:**

```

my\_list.append(6) # Appending an element

my\_list.insert(2, 9) # Inserting at a specific index

print(my\_list.index(4)) # Finding the index of an element

sorted\_list = sorted(my\_list) # Sorting the list

del my\_list[3] # Deleting an element by index

for item in my\_list:

print(item) # Iterating through the list

```

* **Lists of Lists (Matrices):**

```

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(matrix[1][2]) # Accessing an element in a matrix

```

**Tuple**:

* Indexing, Slicing, Creating, Immutability:

 ```

my\_tuple = (1, 2, 3, 4, 5)

print(my\_tuple[0]) # Accessing by index

print(my\_tuple[1:3]) # Slicing

new\_tuple = tuple([6, 7, 8]) # Creating a tuple from a list

```

**- Tuple vs List: Similarities and Differences, Nesting Lists and Tuples:**

- Similarities: Both can store multiple items.

- Differences: Tuples are immutable; lists are mutable.

**Dictionaries:**

- Creating, Indexing, Slicing, Adding/Deleting Keys, Modify Them:

```

my\_dict = {'name': 'John', 'age': 25, 'city': 'New York'}

print(my\_dict['name']) # Accessing by key

my\_dict['job'] = 'Engineer' # Adding a new key-value pair

del my\_dict['age'] # Deleting a key

my\_dict['city'] = 'San Francisco' # Modifying a value

```

- Iterating in a Dictionary - keys(), items(), values():

```

for key in my\_dict.keys():

print(key) # Iterating through keys

for item in my\_dict.items():

print(item) # Iterating through key-value pairs

for value in my\_dict.values():

print(value) # Iterating through values

String:

\ Escape Character, Quotations and ' Symbols in Strings, Multiline String, and Basic Functions:

```

escape\_string = "This is a \"quote\" inside a string."

multiline\_string = '''This is

a multiline

string.'''

```

**Functions and exceptions**

* **Creating Own Functions and Calling It:**

```

def my\_function(param1, param2):

# Code inside the function

my\_function(arg1, arg2) # Calling the function

```

* **Return and Yield Command to Return a Function's Result:**

```

def add\_numbers(a, b):

return a + b

def generate\_numbers():

for i in range(5):

yield i

```

**- None Keyword:**

- `None` is a special constant in Python that represents the absence of a value.

**- Recursion:**

- A function calling itself.

**- Parameter vs Argument:**

- Parameter is a variable in a method definition. Argument is the actual value passed to the function.

**- Positional Keywords and Mixed Argument Passing:**

- Passing arguments by position and using keywords.

**- Basic Parameter Values:**

```

def my\_function(x=1, y=2):

# Code inside the function

```

**- Make List with list() Command:**

```

my\_list = list("hello") # Converts a string to a list of characters

```

**- Try-Except:**

- Used for handling exceptions.

```

try:

# Code that might raise an exception

except SomeException:

# Code to handle the exception

```

**Advanced python**

**- Modules Usage:**

- Modules are files containing Python code. They can be imported and used in other scripts.

**- PIP:**- PIP is the package installer for Python. It is used to install and manage Python packages.