I have prepared documentation for all the Python topics from Chapter 1 to Chapter 9 based on the key keywords from Sir Zia's repository. This documentation helps students memorize and practice the essential points of each topic using the important keywords. The topics covered include:

1. Introduction to Python
2. Data Types
3. Operators, Keywords, and Variables
4. Strings and Casting
5. Control Flow
6. Lists, Tuples, and Dictionaries
7. Sets
8. Modules and Functions
9. Exception Handling

These keywords are presented in a way that allows students to easily recall the critical aspects of each topic and effectively practice them.

📘 01 - Introduction to Python

Keyword	Explanation
Python	High-level, interpreted, general-purpose programming language
Creator	Guido van Rossum
First Release	1991
Current Version	(As of 2023) Python 3.x (usually 3.10+ used)

Interpreted	Code is executed line-by-line by an interpreter
Dynamically Typed	No need to declare variable types
High-level Language	Easy to read and write, closer to human language
Syntax	Clean and simple (e.g., uses indentation instead of braces `{}`)
Script/REPL	Python can be run as scripts `.py` or interactively in a terminal (REPL)
Use Cases	Web development, Data Science, AI/ML, Automation, Scripting, Games, etc.
Popular IDEs	PyCharm, VS Code, Jupyter Notebook, Thonny
File Extension	`.py`
Comments	`# Single-line` and `''' or """ Multi-line """`
print() Function	Used to display output on the screen (`print("Hello")`)
Variables	Containers to store data (`x = 5`)
Indentation	Whitespace (usually 4 spaces) used to define blocks (e.g., in `if`, `for`)

📘 02 - Data Types in Python

Data Type	Description	Example	Function / Keyword
`int`	Integer numbers (positive/negative whole numbers)	`x = 5`, `y = -10`	`type(x)` → `int`

`float`	Decimal numbers / Floating-point values	`pi = 3.14`, `rate = -2.5`	`type(pi)` → `float`
`str` (String)	Text data enclosed in quotes	`name = "Saira"`	`type(name)` → `str`
`bool`	Boolean (True / False) values	`is_valid = True`	`type(is_valid)` → `bool`
`list`	Ordered, mutable collection	`nums = [1, 2, 3]`	`type(nums)` → `list`
`tuple`	Ordered, immutable collection	`coords = (5, 10)`	`type(coords)` → `tuple`
`set`	Unordered, unique values only	`unique_nums = {1, 2, 3}`	`type(unique_nums)` → `set`
`dict`	Key-value pairs / Dictionary	`student = {"name": "Ali", "age": 20}`	`type(student)` → `dict`
`NoneType`	Represents absence of a value	`x = None`	`type(x)` → `NoneType`
`complex`	Complex numbers (a + bj)	`z = 2 + 3j`	`type(z)` → `complex`

🧠 Bonus Tips:

- Use `type()` function to check data type.
- Use `isinstance(variable, type)` to confirm type.
- Lists, Sets, Tuples can be **iterated** using loops.

📘 03 - Operators, Keywords & Variables

🔧 1. Operators in Python

Operator Type	Symbols / Examples	Purpose
Arithmetic	`+`, `-`, `*`, `/`, `//`, `%`, `**`	Addition, Subtraction, etc.
Comparison	`==`, `!=`, `>`, `<`, `>=`, `<=`	Compare values
Assignment	`=`, `+=`, `-=`, `*=`, `/=`, `%=`, `**=`	Assign or update variables
Logical	`and`, `or`, `not`	Combine boolean expressions
Bitwise	`&`, `` ` ``	`, ^, ~, <<, >>` `
Membership	`in`, `not in`	Check if value exists in a sequence
Identity	`is`, `is not`	Check if the same object (memory add.)

🔑 2. Python Keywords (Reserved Words)

Cannot be used as variable names!

False	**True**	**None**	**and**	**or**	**not**
if	else	elif	while	for	break
continue	in	is	return	def	class
try	except	raise	with	as	import
from	pass	lambda	yield	assert	del
global	nonlocal	finally			

🧠 **Total Keywords:** 35+ in Python 3 (can check using: `help("keywords")`)

🧮 3. Variables

Concept	Details / Examples
Definition	Used to store data in memory (`x = 10`)
Naming Rules	Start with letter or underscore, no special chars/spaces
Valid Names	`my_var`, `_value1`, `age`, `totalAmount`
Invalid Names	`1num`, `user-name`, `class` (keyword)
Dynamic Typing	No need to declare type (`x = 5`, `x = "hello"` ← valid)
Multiple Assignment	`a, b = 5, 10` or `x = y = z = 0`
Swapping Values	`a, b = b, a` ← easy way to swap value

📘 04 - Strings & Casting in Python

🔤 1. Strings in Python

Concept	Explanation / Example
Definition	Sequence of characters in quotes (`'Hello'`, `"World"`)
Multiline Strings	Use triple quotes (`'''Text'''` or `"""Text"""`)
String Indexing	`s = "Python"`, `s[0]` → `'P'`, `s[-1]` → `'n'`
String Slicing	`s[0:3]` → `'Pyt'`, `s[2:]` → `'thon'`, `s[:4]` → `'Pyth'`

String Length	`len(s)`
Loop Through String	`for char in s:`
Immutability	Strings cannot be changed → `s[0] = 'X'` ❌ (Error)
Concatenation	`'Hello' + ' World' → 'Hello World'`
Repetition	`'Hi' * 3 → 'HiHiHi'`
String Methods	`lower()`, `upper()`, `title()`, `strip()`, `replace()`, `split()`
f-Strings	`name = 'Saira'`, `f"Hello {name}"`
Escape Characters	`\'`, `\"`, `\\`, `\n`, `\t`
Check Substring	`'th' in 'Python' → True`

🔁 2. String Methods Quick Reference

Method	Description	Example
`str.lower()`	Convert to lowercase	`"HELLO".lower()` → `'hello'`
`str.upper()`	Convert to uppercase	`"hi".upper()` → `'HI'`
`str.title()`	Capitalize each word	`"my name".title()` → `'My Name'`
`str.strip()`	Remove surrounding whitespace	`" hello ".strip()` → `'hello'`
`str.replace()`	Replace characters	`"aabb".replace('a','x')` → `'xxbb'`
`str.split()`	Split into list	`"a,b,c".split(',')` → `['a','b','c']`
`str.find()`	Find index of substring	`"hello".find('e')` → `1`

🔁 3. Type Casting in Python

Casting Type	Function	Example	Result
String to Integer	`int()`	`int("10")`	`10` (int)
String to Float	`float()`	`float("3.14")`	`3.14` (float)
Integer to String	`str()`	`str(25)`	`"25"` (str)
Float to Integer	`int()`	`int(5.99)`	`5`
Integer to Float	`float()`	`float(10)`	`10.0`
Boolean to Integer	`int(True)`		`1`
String to List	`list("abc")`		`['a', 'b', 'c']`

✅ Tips:

- Invalid casting (e.g. `int("abc")`) will cause an error!
- Always validate data before casting if unsure.

📘 05 - Control Flow in Python

🧭 1. Conditional Statements

Statement	Syntax	Purpose
`if`	`if condition:`	Run block if condition is true
`if-else`	`if condition:\n ...\nelse:\n ...`	Choose between two blocks
`if-elif-else`	`if ... elif ... else:`	Multiple conditions check

Comparison Operators	`==`, `!=`, `>`, `<`, `>=`, `<=`	Used within conditions
Logical Operators	`and`, `or`, `not`	Combine multiple conditions

```
age = 20
if age >= 18:
    print("Adult")
else:
    print("Minor")
```

🔁 2. Looping Statements

Loop Type	Syntax / Use	Purpose
`for` loop	`for item in iterable:`	Loop over lists, strings, ranges, etc.
`while` loop	`while condition:`	Loop until condition is false
`break`	Exit loop early	Used inside `for` or `while`
`continue`	Skip current iteration, continue to next	Skip unnecessary step in loop
`range()`	`range(start, stop, step)`	Used with `for` loops
`enumerate()`	Get index and value while looping	`for i, val in enumerate(list):`

🔁 **Examples:**

```
# for loop
for i in range(3):
    print(i)

# while loop
x = 0
while x < 3:
    print(x)
```

```
    x += 1
```

🧠 3. Match-Case (Python 3.10+) — Switch Alternative

Keyword	Syntax / Use	Example
`match-case`	Pattern matching — similar to switch-case in other languages	`match value:\n case ...`

```
status = 200
match status:
    case 200:
        print("OK")
    case 404:
        print("Not Found")
```

✅ Summary Mind Map:

- **Decisions:** `if`, `elif`, `else`
- **Loops:** `for`, `while`, `break`, `continue`
- **Patterns (Python 3.10+):** `match-case`

📘 06 - Lists, Tuples, and Dictionaries in Python

📋 1. Lists (Mutable, Ordered Collection)

Feature	Example	Result / Notes
Create a List	`my_list = [1, 2, 3]`	List of integers

Indexing	`my_list[0]`	`1` (1st element)
Slicing	`my_list[1:]`	`[2, 3]`
Modify Item	`my_list[1] = 5`	Changes 2 to 5
Append	`my_list.append(4)`	Adds 4 at end
Insert	`my_list.insert(1, 9)`	Insert 9 at index 1
Remove	`my_list.remove(2)`	Removes first occurrence of 2
Pop	`my_list.pop()`	Removes last item
Length	`len(my_list)`	Total items in list
Loop	`for i in my_list:`	Loop through elements
Nested Lists	`[[1,2], [3,4]]`	2D list
List Comprehension	`[x**2 for x in range(3)]`	`[0, 1, 4]`
Membership	`5 in my_list`	`True` if found

📦 2. Tuples (Immutable, Ordered Collection)

Feature	**Example**	**Result / Notes**
Create a Tuple	`my_tuple = (1, 2, 3)`	Tuple of integers
Single Item	`one_item = (1,)`	Comma is **required**
Access Item	`my_tuple[0]`	`1`
Immutability	`my_tuple[1] = 5`	❌ Error — can't modify
Slicing	`my_tuple[1:]`	`(2, 3)`
Tuple Packing/Unpacking	`a, b = (10, 20)`	`a = 10, b = 20`

Useful for fixed data	Coordinates: `(x, y)`	Lightweight and safe from changes

---

🧠 3. Dictionary (Key-Value Pair Collection)

Feature	Example	Result / Notes
Create a Dict	`student = {"name": "Saira", "age": 22}`	Key-value pairs
Access Value	`student["name"]`	`'Saira'`
Add/Update Value	`student["city"] = "Karachi"`	Adds new key or updates
Delete Key	`del student["age"]`	Removes `'age'`
Get Method	`student.get("grade", "N/A")`	Avoids KeyError
Keys/Values/Items	`keys()`, `values()`, `items()`	List-like views
Looping	`for k, v in student.items():`	Loop through key-value pairs
Nested Dictionary	`{"user": {"name": "Ali", "age": 30}}`	Access: `dict["user"]["age"]`

✅ Summary Mind Chart

Type	Mutable	Ordered	Indexed	Best For
`List`	✅	✅	✅	General purpose collections
`Tuple`	❌	✅	✅	Fixed data (e.g. coordinates)
`Dictionary`	✅	❌ (Py<3.7)	🔑 Key-based	Structured info (like JSON)

📘 07 - Sets in Python

🔢 1. Basic Set Operations

Feature	Example	Result / Notes
Create a Set	`my_set = {1, 2, 3}`	Set with unique elements
Create from List	`set([1, 2, 2, 3])`	`{1, 2, 3}` (duplicates removed)
Set Length	`len(my_set)`	Returns number of elements
Add Element	`my_set.add(4)`	Adds `4` to the set
Remove Element	`my_set.remove(2)`	Removes `2` (raises error if not found)
Discard Element	`my_set.discard(2)`	Removes `2` (no error if not found)
Pop an Element	`my_set.pop()`	Removes a random element
Clear Set	`my_set.clear()`	Removes all elements

🔀 2. Set Operations (Mathematical)

Operation	Syntax	Result / Example
Union	`A	B` or `A.union(B)`	
Intersection	`A & B` or `A.intersection(B)`	`{1, 2, 3} & {3, 4, 5}` → `{3}`
Difference	`A - B` or `A.difference(B)`	`{1, 2, 3} - {3, 4, 5}` → `{1, 2}`
Symmetric Difference	`A ^ B` or `A.symmetric_difference(B)`	`{1, 2, 3} ^ {3, 4, 5}` → `{1, 2, 4, 5}`

🔁 3. Set Methods

Method	Description	Example
`add()`	Adds an element to the set	`my_set.add(6)` → `{1, 2, 3, 6}`
`remove()`	Removes an element from the set	`my_set.remove(3)` → `{1, 2, 6}`
`discard()`	Removes an element, no error if not found	`my_set.discard(3)` → `{1, 2, 6}`
`pop()`	Removes and returns a random element	`my_set.pop()`
`clear()`	Removes all elements from the set	`my_set.clear()` → `set()`
`copy()`	Returns a copy of the set	`my_set.copy()` → `{1, 2, 3, 6}`
`union()`	Returns the union of two sets	`A.union(B)` → `A
`intersection()`	Returns the intersection of two sets	`A.intersection(B)` → `A & B`
`difference()`	Returns the difference between two sets	`A.difference(B)` → `A - B`
`symmetric_difference()`	Returns the symmetric difference of two sets	`A.symmetric_difference(B)` → `A ^ B`

✅ 4. Key Characteristics of Sets

Characteristic	Detail
Unordered	Elements have no specific order.
No Duplicates	Only unique elements can exist in a set.
Mutable	Elements can be added or removed.
Immutable Elements	Only immutable types (e.g., numbers, strings, tuples) can be added.

Fast Membership Test	Checking if an item exists in a set is very fast.

✅ Quick Summary:

- **Mutable:** Yes
- **No duplicates:** Yes
- **Unordered:** Yes
- **Methods:** `add()`, `remove()`, `union()`, `intersection()`, etc.

📘 08 - Modules and Functions in Python

🔑 1. Functions (Reusable Blocks of Code)

Feature	Example	Result / Notes
Define a Function	`def greet():`	Defines a function called `greet`
Function with Arguments	`def greet(name):`	Function that takes `name` as argument
Return Value	`def add(a, b): return a + b`	Function that returns the sum of `a` and `b`
Call a Function	`greet('Saira')`	Calls the `greet` function with argument `Saira`
Default Arguments	`def greet(name='Guest'):`	Default value for argument `name`
Keyword Arguments	`greet(name='Saira')`	Pass arguments by name
Variable-Length Args	`def add(*args):`	Accepts a variable number of arguments

Lambda Functions	`sum = lambda x, y: x + y`	Anonymous function (short, one-line)

🔍 **Example:**

```
def multiply(x, y):
    return x * y

result = multiply(2, 3)  # Returns 6
```

🔄 2. Modules (Reusable Code Files)

Feature	Example	Result / Notes
Import a Module	`import math`	Import the `math` module
Import Specific Function	`from math import sqrt`	Only imports `sqrt` from `math`
Alias for Module	`import math as m`	Use `m.sqrt()` instead of `math.sqrt()`
Accessing Module Content	`math.pi`	Access `pi` from `math`
Create a Module	`# In mymodule.py def greet():`	Save the function in a file `mymodule.py`
Reload a Module	`import importlib importlib.reload(module)`	Reload module after changes

🔍 **Example:**

```
import math
result = math.sqrt(16)  # Returns 4.0
```

💡 3. Common Built-in Functions

Function	Purpose	Example
`print()`	Output to console	`print("Hello World")`
`len()`	Length of an object	`len("Python")` → `6`
`type()`	Return the type of an object	`type(3)` → `<class 'int'>`
`input()`	Take input from user	`name = input("Enter your name: ")`
`range()`	Generate a sequence of numbers	`range(5)` → `0, 1, 2, 3, 4`
`sum()`	Sum of iterable	`sum([1, 2, 3])` → `6`
`max()`, `min()`	Maximum or minimum of iterable	`max([1, 2, 3])` → `3`, `min([1, 2, 3])` → `1`

⚡ 4. Function Scope and Lifetime

Scope	Description	Example
Global Scope	Variables defined outside a function	Can be accessed from anywhere in the program
Local Scope	Variables defined inside a function	Only accessible within that function
Nonlocal Scope	Access variables in enclosing (but not global) scope	Used in nested functions

---

✅ 5. Summary Comparison: Functions vs Modules

Aspect	Function	Module
Purpose	Reusable block of code	Grouping related code in a separate file
Scope	Local within function call	Global, accessible after importing
Return Type	Can return values	Modules do not return, but provide content

Usage	Used in functions and programs	Used across programs, imported for reuse

---

✅ Quick Summary:

- **Functions:** Reusable, take arguments, return values, can be nested, and can be lambda (anonymous).
- **Modules:** Files containing related functions or classes. Can be imported and reused.

📘 09 - Exception Handling in Python

🛑 1. Basic Exception Handling

Keyword	Usage	Example
`try`	Block to execute code that might raise an error	`try: <code>`
`except`	Catch specific exceptions raised in the try block	`except <ExceptionType>:`
`else`	Execute if no exception occurs	`else: <code>`
`finally`	Always execute, regardless of whether an exception occurs or not	`finally: <code>`

🔍 **Example:**

```
try:
    x = 5 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
finally:
    print("This will always execute.")
```

🧩 2. Common Exception Types

Exception Type	Description	Example
`ZeroDivisionError`	Raised when dividing by zero	`x = 1 / 0`
`ValueError`	Raised when a function receives an invalid argument	`int('hello')`
`IndexError`	Raised when accessing an index that is out of range	`lst = [1, 2]; print(lst[3])`
`KeyError`	Raised when accessing a non-existent key in a dictionary	`d = {'a': 1}; print(d['b'])`
`TypeError`	Raised when an operation is performed on an inappropriate type	`x = '5' + 3`
`FileNotFoundError`	Raised when attempting to open a file that does not exist	`open('nonexistent_file.txt')`

⚡ 3. Catching Multiple Exceptions

Feature	Example	Result
Catch Multiple Exceptions	`except (TypeError, ValueError):`	Catch multiple types of exceptions in one block
Catch All Exceptions	`except Exception as e:`	Catch any exception (avoid in production)

🔍 **Example:**

```
try:
    x = int(input("Enter a number: "))
except (ValueError, TypeError):
    print("Invalid input!")
except Exception as e:
```

```
    print(f"Unexpected error: {e}")
```

🛠️ 4. Raising Exceptions

Keyword	Usage	Example
`raise`	Manually trigger an exception	`raise ValueError("Custom error message")`
`raise` with Error Object	Trigger an exception with a custom error message	`raise Exception("Something went wrong!")`

🔍 **Example:**

```
def check_age(age):
    if age < 18:
        raise ValueError("Age must be at least 18")
    print("Age is valid.")
check_age(15)
```

🔄 5. Custom Exception Classes

Feature	Example	Result
Define Custom Exception	`class MyError(Exception):`	Define a new error type
Custom Error Message	`raise MyError("This is a custom error.")`	Raise your custom error type with a message

🔍 **Example:**

```
class NegativeAgeError(Exception):
    def __init__(self, message="Age cannot be negative"):
        self.message = message
```

```
        super().__init__(self.message)

try:
    age = -1
    if age < 0:
        raise NegativeAgeError
except NegativeAgeError as e:
    print(e)
```

✅ 6. Exception Handling Summary

Type	Description	Example
`try`	Start of code block that might raise an exception	`try: <code>`
`except`	Handle the exception type	`except <ExceptionType>: <handle>`
`else`	Code that runs if no exceptions are raised	`else: <code>`
`finally`	Code that always runs (cleanup)	`finally: <code>`
`raise`	Manually raise an exception	`raise ValueError("Custom message")`

✅ Quick Summary:

- **Try-Except Block:** Use to catch and handle exceptions
- **Common Exceptions:** `ZeroDivisionError`, `ValueError`, `IndexError`, etc.
- **Raise:** You can manually raise exceptions with `raise`
- **Custom Exceptions:** You can create your own error types by inheriting from `Exception`
  `Created by Saira | Slot: Tuesday | Section A | Sir Zia and Sir Arif`