Module 1 & 2

Created by	Karan Maurya
Status	Not started

Topics

Python Cheatsheet: Module 1 + Module 2

(Advanced + Intermediate Friendly)

Module 1: Python Programming Fundamentals

1.1 Variables and Data Types

Core Concepts:

- **Variables** are pointers (references) to objects in memory.
- Python is **dynamically typed**: you don't need to declare types manually.
- Python uses **object mutability**: some types are mutable (lists), others are immutable (strings, ints).

Common Data Types:

Type	Syntax	Mutable?	Example
int	`x = 5`	No	
float	`pi = 3.14`	No	
str	`name = "Alice"`	No	
bool	`flag = True`	No	
list	`nums = [1,2,3]`	Yes	
tuple	`point = (2,3)`	No	
dict	`user = {"id":1}`	Yes	
set	`unique = {1,2,3}`	Yes	

> Advanced Tip:
>
> Use `type(var)` to dynamically check types, and `isinstance(var, datatype)` for safe type checking in complex programs.

```
if isinstance(x, int):
    print("Integer detected!")
```

1.2 Operators and Expressions

Arithmetic Operators

Operator	Use	Example	Result
+	Addition	5 + 2	7

-	Subtraction	5 - 2	3
*	Multiplication	5 ∗ 2	10
/	Division	5 / 2	2.5
//	Floor Division	5 // 2	2
%	Modulus	5 % 2	1
**	Exponentiation	5 ** 2	25

> Best Practice: Always prefer // for integer division when speed is critical.

1.3 Control Structures (if, for, while)

`if-elif-else`

```
x = 5
if x > 0:
    print("Positive")
elif x == 0:
    print("Zero")
else:
    print("Negative")
```

`for` Loop

```
for i in range(5):  # 0 to 4
    print(i)
```

- `range(start, stop, step)` is lazy (memory efficient).
- Convert to list: `list(range(5)) → [0,1,2,3,4]`

`while` Loop

```
count = 0
while count < 3:
    print(count)
    count += 1
```

> Pro Tip: Use else with loops for elegant post-loop actions.
>
> ```
> for i in range(3):
>     print(i)
> else:
>     print("Loop completed successfully")
> ```

Module 2: Specialized Data Structures in Python

2.1 Lists

Creation and Access

```
nums = [10, 20, 30, 40]
print(nums[0])  # Output: 10
print(nums[-1]) # Output: 40 (negative index)
```

Common List Methods

Method	Description	Example
append(x)	Adds element to end	`nums.append(50)`
insert(i, x)	Inserts at index	`nums.insert(1, 15)`
remove(x)	Removes first occurrence	`nums.remove(20)`
pop(i)	Removes & returns element	`nums.pop(2)`
extend(iterable)	Merges another list	`nums.extend([60,70])`
sort()	Sorts list in-place	`nums.sort()`
reverse()	Reverses list in-place	`nums.reverse()`

Advanced List Slicing

```
a = [1,2,3,4,5]
print(a[1:4])   # [2,3,4]
print(a[::-1])  # [5,4,3,2,1] (reverse list)
```

> Speed Tip: Use list comprehensions instead of loops.
>
> ```
> squares = [x*x for x in range(10)]
> ```

2.2 Tuples

- Tuples are **immutable** sequences, optimized for speed and integrity.

```
coordinates = (10, 20)
x, y = coordinates  # Tuple unpacking
```

- **Singleton tuple:**

  ```
  single = (5,)  # must have comma
  ```

- **Why use tuples?**
  - More memory-efficient than lists
  - Safer for fixed collections

2.3 Sets

- **Unordered** collections of **unique** elements.

```
items = {1, 2, 3, 3, 2}
print(items)  # {1, 2, 3}
```

Common Set Operations

Operation	Example	Output
Union	`A	B`
Intersection	`A & B`	Common elements
Difference	`A - B`	Elements in A not in B
Symmetric Difference	`A ^ B`	Elements not common to both

Example:

```
A = {1,2,3}
B = {3,4,5}
print(A & B)  # {3}
```

> Advanced Usage: Sets are highly efficient for membership testing (x in set), much faster than lists.

2.4 Dictionaries

- Key-value pairs (unordered, mutable, fast lookup).

```
student = {"name": "Alice", "age": 20}
print(student["name"])  # Alice
```

Common Dictionary Methods

Method	Description	Example
get(k, default)	Safe key access	`student.get('grade', 'N/A')`
keys()	Return keys view	`student.keys()`
values()	Return values view	`student.values()`
items()	Return key-value pairs	`student.items()`
update(d)	Merge dictionaries	`student.update({"gender":"F"})`
pop(k)	Remove key and return value	`student.pop("age")`

Dictionary Comprehension

```
squares = {x: x*x for x in range(5)}
print(squares)  # {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Summary for Module 1 + 2

Topic	Key Focus	Special Insights
Variables	Reference model	Mutable vs Immutable
Operators	Lazy Evaluation	Use floor division for ints
Lists	Slicing, Comprehension	Avoid using list + `append` inside loops
Tuples	Packing/Unpacking	Fast, safe containers
Sets	Unique items only	Fast membership checks
Dictionaries	Key-Value fast access	Comprehensions for fast dict builds

This concludes Module 1 + 2!

(Structured for quick revision, deep dives, and immediate implementation.)

Would you like me to continue immediately with Module 3 + 4 in the same clean, deep style?

(If yes, I'll make it even slightly more advanced for the next pair.)

Ready to continue?