Python Ultimate Cheat Sheet

Syntax, Data Structures, OOP, and Best Practices

1 Basics & Syntax

Variables & Types *Python is dynamically typed (no need to declare types).*

```
x = 10          # int: Whole numbers
y = 3.14        # float: Decimal numbers
name = "Alice"  # str: Text strings
is_active = True # bool: Logic (True/False)
none_val = None # NoneType: Represents absence
```

Input/Output

```
name = input("Enter name: ") # Always returns str
print(f"Hello, {name}!") # F-String formatting
```

Type Conversion

```
age = int("25")    # String to Int
price = float("19.99") # String to Float
s = str(100)       # Int to String
```

2 Control Flow

Conditional Statements *Decisions based on logic.*

```
if x > 10:
    print("High")
elif x == 10:
    print("Equal")
else:
    print("Low")
```

Loops *Repeating actions.*

```
# For Loop: Iterates over a sequence
for i in range(5): # 0, 1, 2, 3, 4 (stop is excl)
    print(i)

# While Loop: Runs while condition is True
while x > 0:
    x -= 1
```

Control Keywords

- `break`: Exit the loop immediately.
- `continue`: Skip rest of code, go to next loop.
- `pass`: Do nothing (placeholder for future code).

3 Data Structures

List (Mutable) `[1, 2, 3]`

Tuple (Immutable) `(1, 2, 3)`

Set (Unique) `{1, 2, 3}`

Dict (Key:Val) `{'a': 1}`

Quick Selection Guide

- **List**: Ordered collection (e.g., To-do list).
- **Dict**: Key-based lookup (e.g., Phonebook).
- **Set**: Unique items only (e.g., User IDs).
- **Tuple**: Fixed/read-only (e.g., Coordinates).

List (Ordered, Mutable)

```
nums = [1, 2, 3]
nums.append(4)  # Add to end
nums[0] = 99    # Change value
```

Dictionary (Key-Value, Mutable)

```
d = {'name': 'Alice', 'age': 25}
val = d.get('age', 0) # Safe access (default 0)
d.keys(), d.values()  # Access parts
```

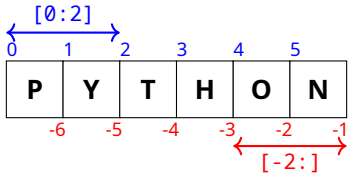
Set (Unordered, Unique)

```
s = {1, 2, 3, 3} # -> {1, 2, 3} (Dups removed)
s.add(4)
```

Tuple (Ordered, Immutable)

```
coords = (10, 20)
# coords[0] = 5  # Error! Cannot change
```

4 String & List Slicing



Syntax: `[start : stop (exclusive) : step]`

```
s = "PYTHON"
s[0:2]   # "PY" (Indices 0, 1)
s[-2:]   # "ON" (Last 2 chars)
s[::-1]  # "NOHTYP" (Reverse string)
```

5 Functions

Definition *Reusable blocks of code.*

```
def greet(name, shout=False):
    msg = f"Hello {name}"
    if shout:
        return msg.upper()
    return msg
```

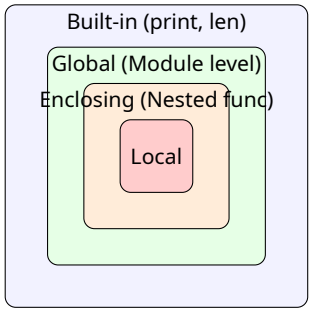
Flexible Arguments

```
def func(*args, **kwargs):
    # *args: Tuple of positional args (1, 2)
    # **kwargs: Dict of keyword args (a=3)
```

Lambda (Anonymous) *One-line functions for short tasks.*

```
square = lambda x: x ** 2
# Usage: map(lambda x: x*2, [1, 2, 3])
```

6 Scope (LEGB Rule)



Python looks for variables in this order: Local → Enclosing → Global → Built-in.

7 Comprehensions

Concise, Pythonic way to create lists/dicts.

List Comprehension

```
# Syntax: [expr for item in iterable if cond]
squares = [x**2 for x in range(10)]
evens = [x for x in nums if x % 2 == 0]
```

Dict Comprehension

```
sq_map = {x: x**2 for x in range(5)}
# {0:0, 1:1, 2:4...}
```

8 Error Handling

Prevent crashes by catching errors.

```
try:
    res = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
except Exception as e:
    print(f"Unknown Error: {e}")
finally:
    print("Always runs (e.g., close DB)")
```

9 File I/O

Context Managers (with) *Automatically handles opening and closing files.*

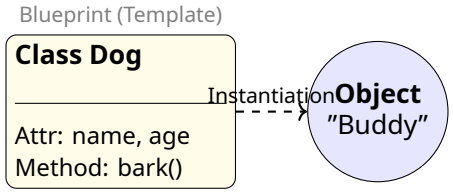
```
# Write Mode ('w' overwrites)
with open("file.txt", "w") as f:
    f.write("Hello World\n")

# Read Mode ('r')
with open("file.txt", "r") as f:
    content = f.read()     # Reads entire file
    lines = f.readlines()  # List of lines
```

Modes

- `'r'`: Read (Default).
- `'w'`: Write (Overwrites file).
- `'a'`: Append (Adds to end).

10 Object-Oriented (OOP)



Classes & Inheritance

```
class Dog:
    # Constructor: Initializes the object
    def __init__(self, name):
        self.name = name  # 'self' refers to instance

    def speak(self): return "Woof"

# Inheritance: Puppy inherits from Dog
class Puppy(Dog):
    def speak(self):    # Polymorphism (Override)
        # super() calls parent method
        return super().speak() + " Yip!"
```

Encapsulation (Private Data) *Hiding data with double underscores.*

```
class Bank:
    def __init__(self):
        self.__bal = 0  # Private attribute

    def deposit(self, amt):
        if amt > 0: self.__bal += amt
```

Abstraction (Interfaces) *Enforcing a structure for child classes.*

```
from abc import ABC, abstractmethod
class Shape(ABC):
    @abstractmethod
    def area(self): pass # Children MUST implement
```

11 Modules & Imports

```
import math
print(math.sqrt(16))

# Import specific function
from os import path
print(path.exists("file.txt"))

# External packages (install via pip)
# pip install pandas
import pandas as pd
```