Python Cheat Sheet

1. Basic Syntax

1.1 Comments

- Single-line: `# This is a comment`
- Multi-line:

```
"""
This is a
multi-line comment
"""
```

1.2 Variables and Assignment

- Variable assignment: `x = 10`
- Multiple assignments: `a, b, c = 1, 2, 3`
- Swap variables: `a, b = b, a`

1.3 Basic Operators

- Arithmetic: `+`, `-`, `*`, `/`, `//` (floor division), `%` (modulus), `**` (exponentiation)

- Comparison: `==`, `!=`, `>`, `<`, `>=`, `<=`
- Logical: `and`, `or`, `not`

---

2. Data Types

2.1 Numeric Types

- Integers: `int` (e.g., `42`)
- Floating-point: `float` (e.g., `3.14`)
- Complex numbers: `complex` (e.g., `3 + 4j`)

2.2 Sequences

- Lists: `list` (e.g., `[1, 2, 3]`)
- Tuples: `tuple` (e.g., `(1, 2, 3)`)
- Strings: `str` (e.g., `"Hello"`)

2.3 Sets

- Sets: `set` (e.g., `{1, 2, 3}`)
- Frozen sets: `frozenset` (e.g., `frozenset({1, 2, 3})`)

2.4 Dictionaries

- Dictionaries: `dict` (e.g., `{'key': 'value'}`)

2.5 Type Conversion

- `int()`, `float()`, `str()`, `list()`, `tuple()`, `set()`, `dict()`

---

3. Control Structures

3.1 Conditional Statements

- `if` statement:

```
if condition:
    # code
elif another_condition:
    # code
else:
    # code
```

3.2 Loops

- `for` loop:

```
for item in iterable:
    # code
```

- `while` loop:

```
while condition:
    # code
```

3.3 Loop Control

- `break`: Exit the loop
- `continue`: Skip the rest of the loop and continue with the next iteration
- `pass`: Do nothing (placeholder)

---

4. Functions

4.1 Defining Functions

- Basic function:

```
def function_name(parameters):
    # code
    return value
```

4.2 Function Arguments

- Positional arguments: `def func(a, b)`
- Keyword arguments: `def func(a=1, b=2)`
- Variable-length arguments:
- `*args`: `def func(*args)`

- ` **kwargs`: `def func(**kwargs)`

4.3 Lambda Functions

- Anonymous functions: `lambda arguments: expression`

```
add = lambda x, y: x + y
```

4.4 Scope

- Local scope: Variables inside a function
- Global scope: Variables outside all functions
- `global` keyword: `global x`

---

5. Modules and Packages

5.1 Importing Modules

- Basic import: `import module_name`
- Import specific functions: `from module_name import function_name`
- Aliasing: `import module_name as alias`

5.2 Creating Modules

- Save functions in a `.py` file and import it.

5.3 Packages

- Directory containing `__init__.py` file.
- Import from package: `from package_name.module_name import function_name`

---

6. File Handling

6.1 Opening Files

- `open(file, mode)`
- Modes: `'r'` (read), `'w'` (write), `'a'` (append), `'b'` (binary)

6.2 Reading Files

- `file.read()`: Read entire file
- `file.readline()`: Read one line
- `file.readlines()`: Read all lines into a list

6.3 Writing Files

- `file.write(string)`: Write string to file
- `file.writelines(list)`: Write list of strings to file

6.4 Closing Files

- `file.close()`

6.5 Context Manager

- `with open(file) as f:`

---

7. Error Handling

7.1 Try-Except Block

- Basic structure:

```
try:
    # code
except ExceptionType as e:
    # handle error
```

7.2 Finally Block

- Always executes:

```
try:
    # code
finally:
    # cleanup code
```

7.3 Custom Exceptions

- Define custom exception:

```
class CustomError(Exception):
    pass
```

---

8. Object-Oriented Programming (OOP)

8.1 Classes and Objects

- Define class:

```
class MyClass:
    def __init__(self, param):
        self.param = param
```

- Create object: `obj = MyClass(value)`

8.2 Methods

- Instance methods: `def method_name(self, args)`
- Class methods: `@classmethod`
- Static methods: `@staticmethod`

8.3 Inheritance

- Define subclass:

```
class SubClass(BaseClass):
    pass
```

8.4 Encapsulation

- Public: `self.attr`
- Protected: `self._attr`
- Private: `self.__attr`

8.5 Polymorphism

- Method overriding: Redefine method in subclass.

---

9. Advanced Topics

9.1 List Comprehensions

- Basic syntax: `[expression for item in iterable if condition]`

```
squares = [x**2 for x in range(10)]
```

9.2 Generators

- Define generator:

```
def my_generator():
    yield value
```

- Use generator: `for item in my_generator():`

9.3 Decorators

- Define decorator:

```
def my_decorator(func):
    def wrapper():
        # code
        func()
        # code
    return wrapper
```

- Use decorator: `@my_decorator`

9.4 Context Managers

- Define context manager:

```
class MyContextManager:
    def __enter__(self):
        # setup code
    def __exit__(self, exc_type, exc_value, traceback):
        # cleanup code
```

---

10. Tips and Tricks

10.1 Help and Documentation

- `help(object)`: Get help on an object
- `dir(object)`: List attributes and methods

10.2 Debugging

- Use `print()` for quick debugging
- Use `pdb` for advanced debugging:

```
import pdb; pdb.set_trace()
```

10.3 Performance

- Use `timeit` module for timing code:

```
import timeit
timeit.timeit('code', number=1000)
```

10.4 Virtual Environments

- Create virtual environment: `python -m venv myenv`
- Activate: `source myenv/bin/activate` (Linux/Mac) or `myenv\Scripts\activate` (Windows)

This cheat sheet provides a comprehensive overview of Python's essential features, syntax, and best practices. Use it as a quick reference to enhance your Python programming skills.

By Ahmed Baheeg Khorshid

ver 1.0