Cheat Sheet for Python with example

Python Cheat Sheet

1. Basic Syntax

1.1 Comments

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

```
"""
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 Print Statements

- Basic print: `print("Hello, World!")`
- Print with variables: `print("Value of x:", x)`

---

2. Data Types

2.1 Numeric Types

- Integer: `x = 10`
- Float: `y = 3.14`
- Complex: `z = 2 + 3j`

2.2 Boolean

- True: `is_true = True`
- False: `is_false = False`

2.3 Strings

- Single-line string: `s = "Hello"`
- Multi-line string:

```
s = """
This is a
multi-line string
"""
```

- String operations:
- Concatenation: `s1 + s2`
- Repetition: `s * 3`
- Indexing: `s[0]`
- Slicing: `s[1:4]`

2.4 Type Conversion

- Convert to integer: `int("10")`
- Convert to float: `float("3.14")`
- Convert to string: `str(10)`

---

3. Control Flow

3.1 Conditional Statements

- `if` statement:

```
if x > 10:
    print("x is greater than 10")
elif x == 10:
    print("x is 10")
else:
    print("x is less than 10")
```

3.2 Loops

- `for` loop:

```
for i in range(5):
    print(i)
```

- `while` loop:

```
while x > 0:
    print(x)
    x -= 1
```

3.3 Loop Control Statements

- `break`: Exit the loop
- `continue`: Skip the current iteration
- `pass`: Do nothing (placeholder)

---

4. Functions

4.1 Defining Functions

- Basic function:

```
def greet(name):
    return "Hello, " + name
```

4.2 Function Arguments

- Positional arguments: `def add(a, b):`
- Default arguments: `def add(a, b=10):`
- Variable-length arguments:
- `*args`: `def func(*args):`

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

4.3 Lambda Functions

- Anonymous function: `add = lambda x, y: x + y`

---

5. Data Structures

5.1 Lists

- Creation: `my_list = [1, 2, 3]`
- Operations:
- Append: `my_list.append(4)`
- Extend: `my_list.extend([5, 6])`
- Insert: `my_list.insert(1, 10)`
- Remove: `my_list.remove(2)`
- Pop: `my_list.pop()`
- Index: `my_list.index(3)`
- Count: `my_list.count(2)`
- Sort: `my_list.sort()`
- Reverse: `my_list.reverse()`

5.2 Tuples

- Creation: `my_tuple = (1, 2, 3)`
- Immutable, but can contain mutable objects

5.3 Sets

- Creation: `my_set = {1, 2, 3}`
- Operations:
- Add: `my_set.add(4)`
- Remove: `my_set.remove(2)`
- Union: `set1 | set2`
- Intersection: `set1 & set2`
- Difference: `set1 - set2`

5.4 Dictionaries

- Creation: `my_dict = {'key1': 'value1', 'key2': 'value2'}`
- Operations:
- Access: `my_dict['key1']`
- Add/Update: `my_dict['key3'] = 'value3'`
- Remove: `del my_dict['key2']`
- Keys: `my_dict.keys()`
- Values: `my_dict.values()`
- Items: `my_dict.items()`

---

6. Modules and Packages

6.1 Importing Modules

- Basic import: `import math`
- Import specific functions: `from math import sqrt`
- Aliasing: `import numpy as np`

6.2 Creating Modules

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

6.3 Packages

- Organize modules in directories with `__init__.py`

---

7. File Handling

7.1 Opening Files

- Open file: `file = open('example.txt', 'r')`
- Modes:

- `'r'`: Read (default)
- `'w'`: Write (truncate)
- `'a'`: Append
- `'b'`: Binary mode

7.2 Reading Files

- Read all: `content = file.read()`
- Read line: `line = file.readline()`
- Read lines: `lines = file.readlines()`

7.3 Writing Files

- Write: `file.write("Hello, World!")`
- Write lines: `file.writelines(["Line1\n", "Line2\n"])`

7.4 Closing Files

- Close file: `file.close()`
- Using `with` statement:

```
with open('example.txt', 'r') as file:
    content = file.read()
```

---

8. Error Handling

8.1 Try-Except Block

- Basic try-except:

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
```

8.2 Multiple Exceptions

- Multiple except blocks:

```
try:
    x = int("a")
except ValueError:
    print("Invalid conversion")
except ZeroDivisionError:
    print("Cannot divide by zero")
```

8.3 Finally Block

- Always executed:

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("Error")
finally:
    print("This will always run")
```

---

9. Object-Oriented Programming (OOP)

9.1 Classes and Objects

- Define class:

```
class Dog:
    def __init__(self, name):
        self.name = name

    def bark(self):
        return "Woof!"
```

- Create object: `my_dog = Dog("Buddy")`

9.2 Inheritance

- Define subclass:

```
class Puppy(Dog):
    def __init__(self, name, age):
        super().__init__(name)
        self.age = age
```

9.3 Encapsulation

- Private attribute: `self.__private_var`

9.4 Polymorphism

- Method overriding:

```
class Puppy(Dog):
    def bark(self):
        return "Yip!"
```

---

10. Advanced Topics

10.1 List Comprehensions

- Basic list comprehension: `squares = [x**2 for x in range(10)]`

10.2 Generators

- Define generator:

```
def my_generator():
    yield 1
    yield 2
    yield 3
```

- Use generator: `gen = my_generator(); next(gen)`

10.3 Decorators

- Define decorator:

```
def my_decorator(func):
    def wrapper():
        print("Before function")
        func()
        print("After function")
    return wrapper
```

- Use decorator:

```
@my_decorator
def say_hello():
    print("Hello!")
```

10.4 Context Managers

- Define context manager:

```
class FileManager:
    def __init__(self, filename, mode):
        self.filename = filename
        self.mode = mode
        self.file = None
```

```
    def __enter__(self):
        self.file = open(self.filename, self.mode)
        return self.file

    def __exit__(self, exc_type, exc_value, exc_traceback):
        self.file.close()
```

- Use context manager:

```
with FileManager('example.txt', 'w') as file:
    file.write("Hello, World!")
```

---

11. Tips and Tricks

11.1 Help and Documentation

- Get help: `help(object)`
- Docstring:

```
def my_function():
    """This is a docstring"""
    pass
```

11.2 Debugging

- Use `print` statements
- Use `pdb` (Python Debugger):

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

11.3 Performance

- Use `timeit` module:

```
import timeit
timeit.timeit('"-".join(str(n) for n in range(100))', number=10000)
```

11.4 Virtual Environments

- Create virtual environment: `python -m venv myenv`
- Activate:

- Windows: `myenv\Scripts\activate`
- Unix/MacOS: `source myenv/bin/activate`

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.