Python Cheat sheet by **Nader Mohamed** OCT 2024

About Me

Hello! I'm **Nader**. I am a passionate **Artificial Intelligence (AI) student** currently studying at **Kafrelsheikh University**. I have a strong interest in **AI**, **machine learning**, **competitive programming**, and exploring the vast world of technology. My journey into the field of programming started with learning `C++`, and I have since expanded my skills into `Python` and `Object-Oriented Programming (OOP)`.

You can find me here : Nader Mohamed

About This Cheat Sheet

Welcome to the **Python Programming Cheat Sheet**! This notebook serves as a comprehensive guide to help you quickly navigate through the fundamental concepts and advanced features of Python programming. Whether you're a beginner starting your journey or an experienced coder looking for quick references, this cheat sheet will be a valuable resource to support your learning and coding efforts.

Date: October 2024

What You Will Find

In this cheat sheet, you'll find:

- **Basic Python Syntax**: Covering variables, data types, conditionals, and loops.
- **Functions**: How to define and call functions, with examples of parameters and return values.
- **Modules and Libraries**: A quick overview of essential libraries like `math`.
- **Advanced Topics**: Covering list comprehensions, lambda functions, decorators, and more.

Why Use This Cheat Sheet?

This cheat sheet is designed to:

- **Save time**: Quickly access the most important Python concepts without digging through documentation.
- **Improve coding skills**: Reinforce key concepts and techniques with simple, easy-to-understand examples.
- **Serve as a reference**: Ideal for competitive programmers, students, or anyone looking for quick answers to coding questions.

How to Use It

1. **Refer to topics**: Each section provides an explanation of Python concepts, followed by clear code examples that you can run directly in the notebook.
2. **Practice the examples**: Experiment with the provided code and modify it to reinforce your understanding.
3. **Explore advanced topics**: Dive into more complex areas of Python, such as OOP, error handling, and efficient coding techniques.

Why Python?

Python is a versatile and powerful language, known for its simplicity and readability. It is widely used in various fields, including:

- **Artificial Intelligence and Machine Learning**
- **Data Science and Data Analysis**
- **Web Development**
- **Automation**
- **Competitive Programming**

This cheat sheet will guide you through Python's core capabilities and get you closer to mastering the language!

Happy Coding!

In [81]:
```python
class NaderMohamed:
    def __init__(self):
        self.name = "Nader Mohamed"
        self.age = 19
        self.university = "Kafrelsheikh University"
        self.faculty = "Faculty of Artificial Intelligence"
        self.interests = ["AI", "Football", "Computers"]
        self.skills = ["C++", "Python", "Microsoft Office", "Object-Orientec

    def introduce(self):
        print(f"Hello! My name is {self.name}.")
        print(f"I am {self.age} years old and currently studying at {self.ur
        print(f"My interests include: {', '.join(self.interests)}.")
        print(f"My skills include: {', '.join(self.skills)}.")
if __name__ == "__main__":
    nader = NaderMohamed()
    nader.introduce()
```

```
Hello! My name is Nader Mohamed.
I am 19 years old and currently studying at Kafrelsheikh University in the F
aculty of Artificial Intelligence.
My interests include: AI, Football, Computers.
My skills include: C++, Python, Microsoft Office, Object-Oriented Programmin
g, Problem Solving.
```

Python Cheat Sheet with Explanations and Examples :-

1. Basic Syntax & Variables

Comments:

Comments are used to explain code or prevent execution.

`# Single-line comment`

`""" Multi-line comment """`

In [88]:
```
# This is a single-line comment
print("Hello, World!")  # This prints 'Hello, World!' to the console

"""
This is a multi-line comment.
It can span multiple lines.
Use it to explain complex logic or provide additional context.
"""
print("Hello, World!")
```

```
Hello, World!
Hello, World!
```

Variables:

Variables store data for processing. Python is dynamically typed, meaning you don’t need to declare the type.

`x = 5            # Integer`

`y = 3.14         # Float`

`name = "Alice"   # String`

`flag = True      # Boolean`

Printing:

The print() function outputs data to the screen.

In [103…
```
print("Hello, World!")          # Output: Hello, World!
print(f"Value of x: {x}")       # Using f-strings to print variables
```

```
Hello, World!
Value of x: 4
```

Escape Sequences:

Escape sequences allow you to include special characters inside strings that are otherwise hard to express.

They begin with a backslash

`(/)`.

Escape Sequence	Meaning	Example
`\\`	Backslash	`"C:\\Program Files\\"`
`\'`	Single Quote	`"It\'s Python!"`
`\"`	Double Quote	`"He said, \"Hello!\""`
`\n`	Newline	`"Hello\nWorld"`
`\t`	Horizontal Tab	`"Python\t3"`
`\r`	Carriage Return	`"123\r456"`
`\b`	Backspace	`"Hello\bWorld"`

In [99]:
```
# Example for Backslash
print("This is a backslash: C:\\Program Files\\")

# Example for Single Quote
print('It\'s a Python example')

# Example for Double Quote
print("He said, \"Hello!\"")

# Example for Newline
print("Hello\nWorld")

# Example for Horizontal Tab
print("Python\tCheat Sheet")

# Example for Carriage Return (moves the cursor back to the start of the lir
print("12345\rABC")  # ABC45 (overwrites first few characters)

# Example for Backspace
print("Hello\bWorld")  # Outputs "HellWorld" (backspace removes the 'o')
```

```
This is a backslash: C:\Program Files\
It's a Python example
He said, "Hello!"
Hello
World
Python	Cheat Sheet
ABC45
HellWorld
```

Indentation in Python:

- Indentation defines code blocks: Python uses indentation to define the structure of loops, functions, conditionals, and classes.

- Standard indentation: Use **4 spaces** for each indentation level.

- Consistent indentation: All lines within a block must be indented the same amount.

Common errors:

- IndentationError: Raised when indentation is incorrect.

- TabError: Raised when you mix spaces and tabs in the same block.

In [112…

```
# Correct Indentation
def my_function():
    if True:
        print("Proper indentation!")
print("Proper indentation!")
print("Don't forget the indentation!")
```

```
Proper indentation!
Don't forget the indentation!
```

2. Data Types & Typecasting

Common Data Types:

int: Integer numbers (whole numbers)

float: Decimal numbers

str: Text or sequence of characters

bool: True/False values

Examples:

```
int_var = 42          # Integer
```

```
float_var = 3.14      # Float
```

```
str_var = "Python"    # String
```

```
bool_var = True       # Boolean
```

Typecasting:

You can convert between types using functions like str(), int(), and float().

`x = str(42)          # Converts integer to string -> "42"`

`y = int("123")       # Converts string to integer -> 123`

`z = float("3.14")    # Converts string to float -> 3.14`

Checking the Type:

Use type() to check the type of a variable.

In [84]:
```
x= "this is a string"
type(x)   # Output: <class 'str'>
```

Out[84]:
```
str
```

3. Operators

Arithmetic Operators:

`x + y   # Addition`

`x - y   # Subtraction`

`x * y   # Multiplication`

`x / y   # Division`

`x // y  # floor division (division without remainder)`

`x ** y  # Exponentiation (power)`

`x % y   # Modulus (remainder)`

In [92]:
```
x = 5
y = 7

print (x + y)
print (x - y)
print (x * y)
print (x / y)
print (x // y)
print (x ** y)
print (x % y)
```

```
12
-2
35
0.7142857142857143
0
78125
5
```

Comparison Operators:

Used to compare values. The result is always a boolean..

`x == y   # Equal to`

`x != y   # Not equal to`

`x > y    # Greater than`

`x < y    # Less than`

`x >= y   # Greater than or equal to`

`x <= y   # Less than or equal to`

`x and y   # True if both are True`

`x or y    # True if at least one is True`

`not x     # Negates the value of x (True becomes False, and vice versa)`

4. Strings

Strings are sequences of characters enclosed in quotes.

`s1 = 'Nader'`

`s2 = "Mohamed"`

`s3 = """Elsayed"""`

Basic Operations:.

In [118…
```python
s = "Hello Nader"
print (s)
print (len(s))          # Length of the string -> 11
print (s.upper())         # Converts to uppercase -> "HELLO NADER"
print (s.lower())       # Converts to lowercase -> "hello nader"
new_s = s.replace("Nader","World")  # "Hello, World!"
print(new_s)
```

```
Hello Nader
11
HELLO NADER
hello nader
Hello World
```

String Slicing:

Slicing allows you to extract a part of a string.

In [122…

```
s = "Nader"
print (s[0])        # First character -> 'N'
print (s[1])        # First character -> 'a'
print (s[-1])       # Last character -> 'r'
print (s[1:4])      # Substring from index 1 to 3 -> 'ade'
print (s[:3])       # First three characters -> 'Nad'
print (s[3:])       # From index 3 to end -> 'er'
```

```
N
a
r
ade
Nad
er
```

String Formatting:

Use f-strings to insert variables into strings:

In [150…

```
age = 30
message = f"I am {age} years old."
print(message)   # Output: I am 20 years old.
```

```
I am 30 years old.
```

5. Lists

Lists are used to store multiple items in a single variable.

Creating & Modifying:

`numbers = [1, 2, 3, 4]`

```
numbers.append(5)          # Adds element to the end -> [1, 2, 3, 4, 5]
```

```
numbers.remove(2)          # Removes the element -> [1, 3, 4, 5]
```

```
numbers[0] = 10            # Modifies the first element -> [10, 3, 4, 5]
```

List Methods:

```
len(numbers)           # Length of list -> 4
```

```
numbers.pop()          # Removes and returns the last element -> [10, 3, 4]
```

```
numbers.sort()         # Sorts the list -> [3, 4, 10]
```

```
numbers.reverse()      # Reverses the list -> [10, 4, 3]
```

List Slicing:

```
numbers[1:3]    # Slice from index 1 to 2 -> [4, 3]
```

```
numbers[:2]     # First two elements -> [10, 4]
```

```
numbers[::2]    # Every second element -> [10, 3]
```

6. Dictionaries

Dictionaries store data in key-value pairs.

Creating & Accessing:

In [171…

```
person = {"name": "John", "age": 30}
print(person["name"])   # Access value -> "John"
person["city"] = "NY"   # Add key-value pair
print(person["city"])   # Access the new added value -> "NY"
```

```
John
NY
```

Dictionary Methods:

In [178…

```
print (person.keys())      # Returns all keys -> dict_keys(['name', 'age', '
print (person.values())    # Returns all values -> dict_values(['John', 30,
print (person.items())     # Returns all key-value pairs
person.get("name") # Safely access value -> "John"
```

```
dict_keys(['name', 'age', 'city'])
dict_values(['John', 30, 'NY'])
dict_items([('name', 'John'), ('age', 30), ('city', 'NY')])
```

Out[178…

```
'John'
```

7. Conditionals

Conditionals control the flow of the program based on certain conditions.

If-else:

In [186…

```
x = 6
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")
```

```
x is greater than 5
```

In [48]:
```
x = 5
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")
```

```
x is equal to 5
```

In [50]:
```
x = 4
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")
```

```
x is less than 5
```

Ternary Operator:

`Short if`

In [191…
```
result = "Positive" if x > 0 else "Negative"
print (result)
```

```
Positive
```

8. Loops

In [195…
```
for i in range(5):
    print(i)        # Output: 0 1 2 3 4

for fruit in ["apple", "banana"]:
    print(fruit)    # Output: apple banana
```

```
0
1
2
3
4
apple
banana
```

While Loop:

While loops repeat as long as a condition is True.

In [198…
```
x = 5
while x > 0:
    print(x)
    x -= 1          # Output: 5 4 3 2 1
```

```
5
4
3
2
1
```

Loop Control:

```
break       # Exits the loop
```

```
continue    # Skips the current iteration
```

9. Functions

Functions are reusable blocks of code that perform a specific task.

Defining & Calling:

In [214…

```python
def greet(name):
    return f"Hello, {name}!"

result = greet("Alice")
print (result)            # Output: Hello, Alice!
```

```
Hello, Alice!
```

Default Arguments:

In [219…

```python
def greet(name="Guest"):
    return f"Hello, {name}!"
print(greet())    # Output: Hello, Guest!
```

```
Hello, Guest!
```

Lambda Functions:

Anonymous functions written in a single line.

In [226…

```python
add = lambda x, y: x + y
print(add(3, 4))  # Output: 7
```

```
7
```

Useful, Common Built-in Functions:

In [232…

```python
print (abs(-5))            # Absolute value -> 5
print (max([1, 2, 3]))     # Maximum value -> 3
print (min([1, 2, 3]))     # Minimum value -> 1
print (sum([1, 2, 3]))     # Sum of elements -> 6
print (round(3.14159, 2))  # Round to 2 decimal places -> 3.14
```

```
5
3
1
6
3.14
```

10. Modules & Importing

There's a lot of modules in python that you can use and we will use `math` as an example

Importing & Using:

In [45]:
```python
import math

# Print the square root of 16
print(math.sqrt(16))  # Output: 4.0

# Print the sine of 30 degrees (converted to radians)
print(math.sin(math.radians(30)))  # Output: 0.5

# Print the value of pi
print(math.pi)  # Output: 3.141592653589793

# Print the factorial of 5
print(math.factorial(5))  # Output: 120

# Print the greatest common divisor of 8 and 12
print(math.gcd(8, 12))  # Output: 4

# Print the natural logarithm `e` of 10
print(math.log(10))  # Output: 2.302585092994046

# Print the base-10 logarithm of 1000
print(math.log10(1000))  # Output: 3.0

# Round down the number 3.7
print(math.floor(3.7))  # Output: 3

# Round up the number 3.3
print(math.ceil(3.3))  # Output: 4
```

```
4.0
0.49999999999999994
3.141592653589793
120
4
2.302585092994046
3.0
3
4
```

Final Thoughts

I’m really happy and proud of this cheat sheet, which is designed as an introductory guide for beginners to get a solid grasp of Python fundamentals. It covers the essential concepts and functions, giving you a strong foundation to build upon.

Remember, this is just the start! Python is an incredibly powerful language, and there's so much more to explore. Keep experimenting with code, try new projects, and dive into the extensive world of Python libraries and modules.

I hope this guide helps you in your journey as you continue learning and growing. Happy coding, and always stay curious!

You can find me here : Nader Mohamed

`Nader </>`