Python Cheat Sheet for Quick Basic Fundamentals

Welcome to your quick reference guide for Python! This cheat sheet covers the fundamental concepts you need to get started with Python programming.

Table of Contents

Basic Syntax

Overview: Learn the foundational rules and structure of Python code, including comments, indentation, and case sensitivity.

- **Comments:** Use `#` for single-line comments and `''' '''` or `""" """` for multi-line comments.

```python
# This is a single-line comment

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

- **Indentation:** Python uses indentation (usually 4 spaces) to define code blocks.

```python
if True:
    print("Indented block")
```

- **Case Sensitivity:** Python is case-sensitive (`Variable` and `variable` are different).

Variables & Data Types

Overview: Understand how to store and manipulate data using variables and different data types in Python.

- **Variables:** Assign values using `=`.

  ```
  x = 5
  name = "Alice"
  ```

- **Data Types:**
  - **Numeric:** `int`, `float`, `complex`
  - **String:** `"Hello"`
  - **Boolean:** `True`, `False`
  - **None:** `None`

- **Type Checking and Conversion:**

  ```
  type(x)              # Check type
  int("10")           # Convert to integer
  str(10)             # Convert to string
  float(5)            # Convert to float
  ```

Operators

Overview: Utilize various operators to perform operations on variables and values, including arithmetic, comparison, logical, and more.

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

  ```
  result = 3 + 2 * 5  # 13
  ```

- **Comparison Operators:** `==`, `!=`, `>`, `<`, `>=`, `<=`

  ```
  5 > 3  # True
  ```

- **Logical Operators:** `and`, `or`, `not`

  ```
  True and False  # False
  ```

- **Assignment Operators:** `=`, `+=`, `-=`, `*=`, `/=`

  ```
  x = 5
  x += 3  # x is now 8
  ```

- **Membership Operators:** `in`, `not in`

```
"a" in "apple"  # True
```

- **Identity Operators:** `is`, `is not`

```
a is b
```

Control Structures

Overview: Control the flow of your programs using conditional statements and loops to execute code blocks based on conditions or repeatedly.

- **If Statement:**

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

- **For Loops:**

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

- **While Loops:**

```
while condition:
    # code
```

- **Break and Continue:**

```
for i in range(10):
    if i == 5:
        break  # Exit loop
    if i % 2 == 0:
        continue  # Skip to next iteration
    print(i)
```

Functions

Overview: Encapsulate reusable code blocks with functions, including defining, calling, and using advanced features like default and lambda functions.

- **Defining Functions:**

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

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

- **Default Parameters:**

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

greet()  # Hello, World!
```

- **Lambda Functions:**

```
add = lambda a, b: a + b
print(add(2, 3))  # 5
```

Data Structures

Overview: Manage and organize data efficiently using Python's built-in data structures like lists, tuples, dictionaries, and sets.

- **Lists:** Ordered, mutable collection.

```
fruits = ["apple", "banana", "cherry"]
fruits.append("date")
print(fruits[1])  # banana
```

- **Tuples:** Ordered, immutable collection.

```
coordinates = (10, 20)
print(coordinates[0])  # 10
```

- **Dictionaries:** Key-value pairs.

```
person = {"name": "Alice", "age": 25}
print(person["name"])  # Alice
person["age"] = 26
```

- **Sets:** Unordered, unique elements.

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

Input and Output

Overview: Handle user interaction by reading inputs and displaying outputs through the console.

- **Printing to Console:**

```
print("Hello, World!")
```

- **Reading Input:**

```
name = input("Enter your name: ")
print(f"Hello, {name}!")
```

Exception Handling

Overview: Manage errors gracefully using try-except blocks to handle exceptions and ensure your program doesn't crash unexpectedly.

- **Try-Except Block:**

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
finally:
    print("Execution completed.")
```

- **Raising Exceptions:**

```
if x < 0:
    raise ValueError("Negative value not allowed")
```

Modules and Packages

Overview: Organize your code and reuse functionality by importing modules and packages, and learn how to install external packages.

- **Importing Modules:**

```
import math
print(math.sqrt(16))  # 4.0
```

- **From Import:**

```
from math import pi
print(pi)  # 3.141592653589793
```

- **Installing Packages (using pip):**

```
pip install package_name
```

Common Built-in Functions

Overview: Utilize Python's built-in functions to perform common tasks efficiently, such as iterating, type conversion, and more.

- **Range:**

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

- **len():**

```
len("Hello")  # 5
len([1, 2, 3])  # 3
```

- **type():**

```python
type(123)  # <class 'int'>
```

- **str(), int(), float():**

```python
str(100)        # "100"
int("10")       # 10
float("3.14")   # 3.14
```

- **print():**

```python
print("Hello", "World", sep="-")  # Hello-World
```

- **input():**

```python
user_input = input("Enter something: ")
```

Tips for Beginners

- **Practice Regularly:** Consistent coding helps reinforce concepts.
- **Read Documentation:** The official Python documentation is a valuable resource.
- **Use Meaningful Variable Names:** Enhances code readability.
- **Keep Code Simple:** Focus on writing clear and understandable code.
- **Debugging:** Use print statements or debugging tools to troubleshoot your code.

Happy Coding!