# Python Cheat Sheet

## Introduction to Python

### - Introduction

- Python is a high-level, interpreted programming language with dynamic semantics.
- **print("hello world")** → prints the message "hello world"

### - Comments

- **Single Line Comments** - created with a hash symbol #.

```
# this is a single-line comment
```

- **Multi-Line Comments** - created with triple quotes ".

```
"This is a multi-line comment.
It spans multiple lines.  "
```

### - Arithmetic Operators

- Let's take as an example **a = 10** and **b = 20**:

| Operators | Description | Example |
|---|---|---|
| + | Addition | `a + b # Returns: 30` |
| - | Subtraction | `a - b # Returns: -10` |
| * | Multiplication | `a * b # Returns: 200` |
| / | Division | `b / a # Returns: 2.0` |
| % | Modulo | `a % b # Returns: 10` |
| ** | Exponential | `a ** b # Returns:` $10^{20}$ |
| // | Floor Division | `9 // 2 # Returns: 4` |

### - String Interpolation

- **String Interpolation:** Inserting variables into strings using f-strings.

```
name = "Alice"
greeting = f"Hello, my name is name."
print(greeting) # Output:  Hello, my name is Alice
```

### - Assignment Operators

- An assignment operator assigns a value to its left operand based on the value of its right operand. Let's take an example **a = 10**

| Operator | Description | Example |
|---|---|---|
| += | Addition Assignment | `a += 3 # a is now 13` |
| -= | Subtraction Assignment | `a -= 5 # a is now 5` |
| *= | Multiplication Assignment | `a *= 2 # a is now 20` |
| /= | Division Assignment | `a /= 4 # a is now 2.5` |

### - Logical Operators

- It takes two values (conditions) and returns a boolean depending on the operator.

| A | B | A *and* B | A *or* B | not A |
|---|---|---|---|---|
| True | True | True | True | False |
| True | False | False | True | False |
| False | True | False | True | True |
| False | False | False | False | True |

## Variable and Data Types

### - Variable

- Variables are used to store data.

### - Declare Variable

- **Variable Declaration**

```
age = 23 # age assigned a value of 23
print(age) # prints 23
```

- **Variable Reassignment**

```
height = 173
height = 200 # height reassigned value 200
print(height) #prints 200
```

### - Scope

Scope defines where variables and functions can be accessed.

- **Global scope** → a value/function that can be used anywhere in the entire program.

```
global_var = "I am global"
def check_global_scope():
    print(global_var) # accessible here
check_global_scope()
print(global_var) # accessible here too
```

- **Local Scope** → variables declared inside a function are local to that function.

```
def my_function():
    local_var = 20
    print(local_var) # accessible here
my_function()
print(local_var) # NameError:  name 'local_var' is not defined
```

## Data Types

### - Type Checking

- Use **type()** to check the data type in Python

### - Numeric Data Types

- **Integers** → represents whole numbers, both positive and negative

```
num1 = 42
num2 = -3
```

- **Float** → represents floating-point numbers (decimal numbers)

```
pi = 3.14
neg_float = -0.001
```

- **Complex** → represents complex numbers, which have a real part and an imaginary part

```
num1 = 2 + 3j
num2 = -1j
```

# Python Cheat Sheet

## Data Types Continued

### - String Type

- **str** → represents a sequence of characters

```
name = "John Doe"
print(type(name)) #prints <class 'str'>
```

### - Sequence Types

- **List** → represents an ordered mutable sequence

```
fruits = ["apple", "banana", "cherry"]
print(type(fruits)) # prints <class 'list'>
```

- **Tuple** → represents an ordered immutable sequence

```
coordinates = (10, 20)
print(type(coordinates)) # prints <class 'tuple'>
```

- **Range** → represents an immutable sequence of numbers

```
numbers = range(5)
print(type(numbers)) # prints <class 'range'>
```

### - Boolean Type

- **bool** → represents true or false values

```
is_true = True
print(type(is_true)) # prints <class 'bool'>
```

### - Mapping Type

- **dict** → represents a collection of key-value pairs

```
person = {"first_name": "John", "last_name": "Doe"}
print(type(person)) # prints <class 'dict'>
```

### - Set Types

- **set** → represents an unordered collection of unique elements

```
unique_numbers = 1, 2, 3
print(type(unique_numbers)) # prints <class 'set'>
```

- **frozenset** → represents an immutable set

```
frozen_numbers = frozenset([1, 2, 3])
print(type(frozen_numbers)) # prints <class 'frozenset'>
```

### - None Type

- **NoneType** → represents the absence of a value

```
value = None
print(type(value)) # prints <class 'NoneType'>
```

## Lists

- Lists are ordered collections of items.

### - Manipulating a list

- **Creating a List**

```
fruits = ["apple", "banana", "cherry"]
```

- **Accessing Elements**

```
print(fruits[0]) # Output: apple
print(fruits[-1]) # Output: cherry
```

- **Slicing a List**

```
print(fruits[1:3]) # Output: ['banana', 'cherry']
```

- **Adding Elements**

```
fruits.append("date")
fruits.insert(1, "blueberry")
print(fruits)
# Output: ['apple', 'blueberry', 'banana', 'cherry', 'date']
```

- **Removing Elements**

```
fruits.remove("banana")
popped_fruit = fruits.pop()
print(fruits) # Output: ['apple', 'blueberry', 'cherry']
print(popped_fruit) Output: date
```

## Dictionaries

- Dictionaries are collections of key-value pairs.

### - Manipulating a dictionary

- **Creating a Dictionary**

```
person = {
    "first_name": "John",
    "last_name": "Doe",
    "age": 30
}
```

- **Accessing Values**

```
print(person["first_name"]) # Output: John
```

- **Adding/Updating Key-Value Pairs**

```
person["email"] = "john.doe@example.com"
person["age"] = 31
print(person)
# Output: 'first_name': 'John', 'last_name': 'Doe', 'age': 31,
'email': 'john.doe@example.com'
```

- **Removing Key-Value Pairs**

```
del person["email"]
print(person)
# Output: 'first_name': 'John', 'last_name': 'Doe', 'age': 31
```

# Python Cheat Sheet

## Control Flow

Control flow statements control the flow of execution in a program.

### - Comparison Operators

- Comparison operators compare two values and return a boolean.

| Operator | Description | Example |
|---|---|---|
| == | Equal | `a == b # Returns: False` |
| != | Not equal | `a != b # Returns: True` |
| > | Greater than | `a > b # Returns: False` |
| < | Less than | `a < b # Returns: True` |
| >= | Greater than or equal to | `a >= b # Returns: False` |
| <= | Less than or equal to | `a <= b # Returns: True` |

### - If-Else Statement

- **If Statement** → Executes a block of code if a condition is true.

```
x = 10
if x > 5:
    print("x is greater than 5")
```

- **If-Else Statement** → Executes a block of code if a condition is true, otherwise executes another block of code.

```
x = 10
if x > 5:
    print("x is greater than 5")
else:
    print("x is not greater than 5")
```

- **Elif Statement** → Executes a block of code if a condition is true, otherwise checks the next condition.

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

## Loops

- **For Loop** → Iterates over a sequence of items.

```
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    print(num) #ouput:  1, 2, 3, 4, 5
```

- **While Loop** → Repeats a block of code as long as a condition is true.

```
count = 0
while count < 5:
    print(count)
    count += 1
#output:  0, 1, 2, 3, 4
```

## Functions

Functions are blocks of code that perform a specific task and can be reused.

- **Defining a Function**

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

- **Default Parameters**

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

- **Keyword Arguments**

```
def greet(name, greeting="Hello"):
    return f"{greeting}, {name}!"
print(greet(name="Alice", greeting="Hi"))
# Output:  Hi, Alice!
```

## Classes and Objects

**Classes** are blueprints for creating objects. **Objects** are instances of classes.

- **Defining a Class**

```
class Person:
    #constructor for the class
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name
    # defining a method in class
    def greet(self):
        return f"Hello, {self.first_name} {self.last_name}!"
```

- **Creating an Object**

```
# instance of a class (object)
john = Person("John", "Doe")
print(john.greet())
# Output:  Hello, John Doe
```

- **Accessing Object Attributes**

```
print(john.first_name)
# Output:  John
```

- **Modifying Object Attributes**

```
john.first_name = "Jane"
print(john.greet())
# Output:  Hello, Jane Doe
```