Python Basics

1. Syntax

Python's syntax is designed to be readable and straightforward, which makes it an excellent language for beginners. Key features include:

Indentation: Python uses indentation to define blocks of code (e.g., loops, functions, classes). Unlike other languages that use braces `{}` or keywords, Python requires consistent indentation.
python

```
if x > 0:
    print("Positive")
else:
    print("Non-positive")
```

-

Comments: Single-line comments start with a `#`, while multi-line comments can be done using triple quotes `"""` or `'''`.
python

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

-

Statements: Python typically uses one statement per line, but you can use a semicolon `;` to separate multiple statements on a single line.
python

```
x = 5; y = 10; z = x + y
```

-

2. Data Types

Python has several built-in data types, which can be broadly categorized into:

- **Numeric Types**:
    - `int`: Integer values
    - `float`: Floating-point numbers
    - `complex`: Complex numbers

python

```
a = 5       # int
b = 3.14    # float
c = 2 + 3j  # complex
```

-
- **Sequence Types**:
    - `str`: Strings (text)
    - `list`: Ordered, mutable collections
    - `tuple`: Ordered, immutable collections

python

```
s = "Hello, World!"  # str
l = [1, 2, 3, 4]     # list
t = (1, 2, 3, 4)     # tuple
```

-
- **Mapping Type**:
    - `dict`: Key-value pairs, like a hash map or dictionary

python

```
d = {"name": "Alice", "age": 25}  # dict
```

-
- **Set Types**:
    - `set`: Unordered collection of unique elements
    - `frozenset`: Immutable set

python

```
s = {1, 2, 3, 4}           # set
fs = frozenset([1, 2, 3])  # frozenset
```

-
- **Boolean Type**:
    - `bool`: Represents `True` or `False`

python

```
b = True  # bool
```

-
- **None Type**:
    - `None`: Represents the absence of a value or a null value

python

```
n = None  # NoneType
```

-

3. Control Flow

Control flow statements determine the order in which the code executes:

Conditional Statements: `if`, `elif`, and `else`
python

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

-
- **Loops**: `for` and `while`

`for` loop: Iterates over a sequence (like a list, tuple, or string)
python

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

    -

`while` loop: Continues as long as a condition is true
python

```
i = 0
while i < 5:
    print(i)
    i += 1
```

  -

- **Control Statements**: `break`, `continue`, and `pass`
  - `break`: Exits the loop prematurely
  - `continue`: Skips the rest of the code inside the loop for the current iteration and moves to the next iteration
  - `pass`: Does nothing; it's a placeholder

python

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

-

4. Functions

Functions are reusable blocks of code that perform a specific task. In Python, they are defined using the `def` keyword:

Defining a Function:
python

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

-

Calling a Function:
python

```
print(greet("Alice"))
```

-

Default Arguments:
python

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

-
- **Variable-Length Arguments**: Using `*args` and `**kwargs`
    - `*args` allows you to pass a variable number of positional arguments
    - `**kwargs` allows you to pass a variable number of keyword arguments

python

```
def print_args(*args):
    for arg in args:
        print(arg)

def print_kwargs(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")
```

-

Summary

- **Syntax**: Focus on indentation and readability.
- **Data Types**: Understand Python's built-in types.
- **Control Flow**: Master conditionals, loops, and control statements.
- **Functions**: Learn how to define, call, and use functions effectively.