Python Basics

```
# Intro
print("Hello World")

# Getting Started
import sys
print(sys.version)

# Syntax
if 5 > 2:
   print("Five is greater than two!")
```

```
Hello World
3.11.10 (main, Sep  7 2024, 01:03:31) [GCC 13.3.0]
Five is greater than two!
```

Variables and type()

```
x = 5
y = "John"
print(x)
print(y)
print(type(x))
print(type(y))

# Variable Casting
x = str(3)
y = int(3)
z = float(3)

# Variable names
# JavaScript uses Camel Case, myVaribaleName
# Python uses Snake Case my_variable_name

# Mutli variable with values
x, y, z = "Orange", "Blue", "Cherry"
print(x)
print(y)
print(z)

x = y = z = "Orange"
print(x)
print(y)
print(z)
```

```
5
John
<class 'int'>
<class 'str'>
Orange
Blue
Cherry
Orange
Orange
Orange
```

```
# python uses lists over arrays
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

```
apple
banana
cherry
```

```
#python uses print and allows multiple
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)

# Also uses concatenation though does not add
the space
print(x + y + z)

# The concatenation operator does something
different dependent on the data passed to it.
x = 5
y = 10
print(x + y)
```

```
Python is awesome
Pythonisawesome
15
```

```
# Local and global variable, Scope
x = "awesome" # Global Variable

def myfunc():
   x = "fantastic" # Local variable
   # global x = "fantastic" Global keyword
   print("Python is " + x)

myfunc()

print("Python is " + x)
```

```
Python is fantastic
Python is awesome
```

```
# Random number, inclusive low boundary, exclusive upper boundary
import random

print(random.randrange(1, 10))
```

```
9
```

Data Types

str, int, float, complex, list, tuple, range, dict, set, frozenset, bool, bytes, bytearray, memoryview

```
# Python number
x = 1    # int
y = 2.8  # float
z = 1j   # complex
print(type(x))
print(type(y))
print(type(z))

# Data type conversion
x = 1    # int
y = 2.8  # float
z = 1j   # complex

# Convert from int to float:
a = float(x)

# Convert from float to int:
b = int(y)

# Convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

Strings in Python

```
# Defining a string
my_string = "Hello, Python!"
print(my_string)

# Accessing characters in a string
print(my_string[0])   # First character
print(my_string[-1])  # Last character

# String slicing
print(my_string[0:5])   # From index 0 to 4 (Hello)
print(my_string[7:])    # From index 7 to end (Python!)

# String length
print(len(my_string))   # Length of the string

# String Methods
print(my_string.lower())  # Converts to lowercase
print(my_string.upper())  # Converts to uppercase
print(my_string.replace("Python", "World"))  # Replaces "Python" with "World"
print(my_string.split(","))  # Splits string at the comma

# String concatenation
greeting = "Hello"
name = "Alice"
message = greeting + " " + name  # Concatenate with space
print(message)

# String formatting using f-strings (Python 3.6+)
age = 25
print(f"{name} is {age} years old")

# Multiline strings
multi_line_string = """This is a
multiline string
in Python."""
print(multi_line_string)

# String with escape characters
escaped_string = "This is a \"quoted\" word"
print(escaped_string)
```

```
Hello, Python!
H
!
Hello
Python!
14
hello, python!
HELLO, PYTHON!
Hello, World!
['Hello', ' Python!']
Hello Alice
Alice is 25 years old
This is a
multiline string
in Python.
This is a "quoted" word
```

Booleans and Operators in Python

```
#

# Boolean values
x = True
y = False

print(x)
print(y)

# Using Boolean values in conditional statements
if x:
   print("x is True")
else:
   print("x is False")

# Comparison Operators
a = 10
b = 20

print(a == b)  # Equal to (False)
print(a != b)  # Not equal to (True)
print(a > b)   # Greater than (False)
print(a < b)   # Less than (True)
print(a >= b)  # Greater than or equal to (False)
print(a <= b)  # Less than or equal to (True)

# Logical Operators
print(x and y)  # AND operator
print(x or y)   # OR operator
print(not x)    # NOT operator

# Identity Operators
# is and is not checks if two variables point to the same object
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is z)  # True: both point to the same object
print(x is not y)  # True: x and y point to different objects
```

```
True
False
x is True
False
True
False
True
False
True
False
True
False
True
False
True
True
```

```
# Membership Operators
# in and not in check if a value exists within a sequence (e.g., list, tuple,
string)
fruits = ["apple", "banana", "cherry"]

print("apple" in fruits)  # True: "apple" is in the fruits list
print("orange" not in fruits)  # True: "orange" is not in the fruits list

# Bitwise Operators
# These operate on binary representations of numbers
x = 5  # Binary: 0101
y = 3  # Binary: 0011

print(x & y)  # AND operator: 0101 & 0011 = 0001 (1)
print(x | y)  # OR operator: 0101 | 0011 = 0111 (7)
print(x ^ y)  # XOR operator: 0101 ^ 0011 = 0110 (6)
print(~x)     # NOT operator: Inverts all the bits of x, result: -6
print(x << 1) # Left shift: Shifts bits to the left by 1, result: 10
print(x >> 1) # Right shift: Shifts bits to the right by 1, result: 2
```

```
True
True
1
7
6
-6
10
2
```

Lists, Tuples, Sets, and Dictionaries in Python

```
# Lists
fruits = ["apple", "banana", "cherry"]
print(fruits)  # Output: ['apple', 'banana', 'cherry']

# Accessing elements
print(fruits[0])
print(fruits[-1])

# Slicing a list
print(fruits[1:3])

# Adding elements to a list
fruits.append("orange")
print(fruits)

# Removing elements from a list
fruits.remove("banana")
print(fruits)

# Length of a list
print(len(fruits))
```

```
['apple', 'banana', 'cherry']
apple
cherry
['banana', 'cherry']
['apple', 'banana', 'cherry', 'orange']
['apple', 'cherry', 'orange']
3
```

```
# Tuples
coordinates = (4, 5)
print(coordinates)

# Accessing elements
print(coordinates[0])

# Length of a tuple
print(len(coordinates))
```

```
(4, 5)
4
2
```

```python
# Sets
colors = {"red", "green", "blue"}
print(colors)

# Adding elements to a set
colors.add("yellow")
print(colors)

# Removing elements from a set
colors.remove("green")
print(colors)

# Checking membership
print("red" in colors)
print("green" in colors)
```

```
{'blue', 'green', 'red'}
{'blue', 'green', 'red', 'yellow'}
{'blue', 'red', 'yellow'}
True
False
```

Dictionaries

```
person = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

print(person)

# Accessing values
print(person["name"])

# Modifying values
person["age"] = 31
print(person)

# Adding a new key-value pair
person["job"] = "Engineer"
print(person)

# Removing key-value pairs
del person["city"]
print(person)

# Length of a dictionary
print(len(person)

# Looping through a dictionary
for key, value in person.items():
    print(key, ":", value)
```

```
{'name': 'John', 'age': 30, 'city': 'New York'}
John
{'name': 'John', 'age': 31, 'city': 'New York'}
{'name': 'John', 'age': 31, 'city': 'New York', 'job': 'Engineer'}
{'name': 'John', 'age': 31, 'job': 'Engineer'}
3
name : John
age : 31
job : Engineer
```

If-Else Statements in Python

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

# Elif (else if) condition
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")

# Nested if-else
x = 10
y = 20
if x > 5:
    if y > 10:
        print("x is greater than 5 and y is greater than 10")
    else:
        print("x is greater than 5 but y is not greater than 10")
else:
    print("x is not greater than 5")

# Using logical operators in if-else
x = 10
y = 20
if x > 5 and y > 10:
    print("x is greater than 5 and y is greater than 10

# Checking multiple conditions
age = 20
if age < 18:
    print("You are a minor")
elif age >= 18 and age < 21:
    print("You are a young adult")
else:
    print("You are an adult")
```

```
x is greater than 5
x is equal to 5
x is greater than 5 and y is greater than 10
x is greater than 5 and y is greater than 10
You are a young adult
```

While Loop

A while loop in Python repeatedly executes a block of code if the condition is True.

```
i = 1
while i < 6:
    print(i)  # This will print numbers from 1 to 5
    i += 1  # Increment i by 1 after each iteration

# Break and Continue in While Loops
# Break: Terminates the loop
i = 1
while i < 6:
    if i == 3:
      break  # Exit the loop when i equals 3
    print(i)
    i += 1

# Continue: Skips the current iteration and moves to the next
i = 1
while i < 6:
    i += 1
    if i == 3:
      continue  # Skip when i equals 3
    print(i)
```

```
1
2
3
4
5
1
2
2
4
5
6
```

For Loop

A for loop in Python is used to iterate over a sequence (list, string, range).

```
# Example 1: Looping through a list
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
                print(fruit)  # This will print each fruit in the list

# Example 2: Looping through a range of numbers
for x in range(6):  # range(6) gives numbers from 0 to 5
                print(x)  # This will print numbers from 0 to 5

# Nested Loops in Python
# Example of a nested for loop: A loop inside another loop
for x in range(3):  # Outer loop
                for y in range(2):  # Inner loop
                                print(f"({x}, {y})")  # This will print pairs
of x and y

# For Loop with Else
# The else block in a for loop runs when the loop finishes without hitting a
break statement.
for x in range(6):
                print(x)
else:
                print("Loop finished")  # This will print once the loop is
completed
```

```
apple
banana
cherry
0
1
2
3
4
5
(0, 0)
(0, 1)
(1, 0)
(1, 1)
(2, 0)
(2, 1)
0
1
2
3
4
5
Loop finished
```

Functions in Python

```
# Defining a function
def greet(name):
    print("Hello, " + name + "!")

# Calling a function
greet("John")  # Output: Hello, John!

# Function with multiple parameters
def add_numbers(a, b):
    return a + b

result = add_numbers(3, 5)  # Calling the function with arguments
print(result)  # Output: 8

# Function with default parameter value
def greet_person(name="Guest"):
    print("Hello, " + name + "!")

greet_person()  # Output: Hello, Guest!
greet_person("Alice")  # Output: Hello, Alice!

# Function with keyword arguments
def person_info(name, age):
    print("Name: " + name)
    print("Age: " + str(age))

# Calling the function with keyword arguments
person_info(age=30, name="Bob")  # Output: Name: Bob, Age: 30

# Function with arbitrary number of arguments (*args)
def print_fruits(*fruits):
    for fruit in fruits:
        print(fruit)

# Calling the function with multiple arguments
print_fruits("Apple", "Banana", "Cherry")

# Function with arbitrary keyword arguments (**kwargs)
def display_info(**info):
    for key, value in info.items():
        print(key + ": " + str(value))

# Calling the function with multiple keyword arguments
```

```
Hello, John!
8
Hello, Guest!
Hello, Alice!
Name: Bob
Age: 30
Apple
Banana
Cherry
name: Tom
age: 25
city: New York
20
First argument: Hello
Additional arguments: ('World', 42)
Keyword arguments: {'country': 'USA', 'age': 30}
```

```
display_info(name="Tom", age=25, city="New York")

# Return statement in a function
def multiply(x, y):
    return x * y

result = multiply(4, 5)  # Calling the function and storing the result
print(result)  #

# Function with a variable number of arguments (both *args and **kwargs)
def mixed_arguments(arg1, *args, **kwargs):
    print("First argument:", arg1)
    print("Additional arguments:", args)
    print("Keyword arguments:", kwargs)

# Calling the function with both args and kwargs
mixed_arguments("Hello", "World", 42, country="USA", age=30)
```

Lambda Functions in Python

```
# Basic Lambda Function

# A simple lambda function that adds 10 to the input value
add_ten = lambda x: x + 10
print(add_ten(5))

# Lambda Function with Multiple Arguments
# A lambda function that multiplies two numbers
multiply = lambda x, y: x * y
print(multiply(3, 4))  # Output: 12

# Lambda with map()
numbers = [1, 2, 3, 4, 5]

# Squaring each number in the list using a lambda function
squared_numbers = list(map(lambda x: x ** 2, numbers))
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]

# Lambda with filter()
numbers = [1, 2, 3, 4, 5, 6]

# Filtering out the even numbers
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # Output: [2, 4, 6]
```

```
15
12
[1, 4, 9, 16, 25]
[2, 4, 6]
24
5
[(3, 1), (1, 2), (4, 3), (5, 4)]
```