

Fundamentals of Python

PYTHON BASICS



Shruti Bharat

@shrutibharat0105



Table of Contents:

1. Data Types
2. Variables
3. Keywords and identifiers
4. Operators
5. Conditional statements
6. Loops
7. Loop control statements
8. Bonus tips



Data Types

Data types in Python classify the kinds of values that can be stored and manipulated within a program. They define the operations that can be performed on the data and the way it is stored.

```
Basic.py

# Numeric Data Types
integer_num = 10
float_num = 3.14
complex_num = 2 + 3j

# Sequence Data Types
my_list = [1, 2, 3, 4, 5]
my_tuple = (1, 2, 3, 4, 5)
my_set = {1, 2, 3, 4, 5}
my_frozenset = frozenset({1, 2, 3, 4, 5})
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}

# Boolean Data Type
is_active = True

# None Type
no_value = None

# String Data Type
my_string = "Hello, World!"
```

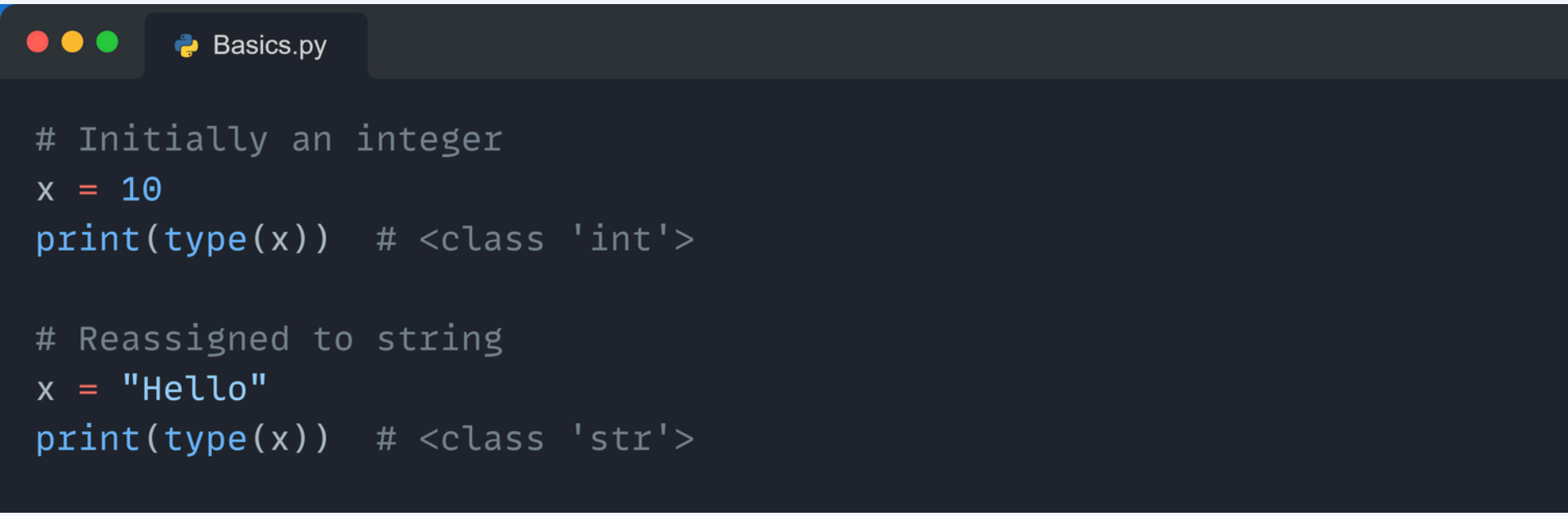


Variables: The Building Blocks

Variables are like containers. They hold data that can change as the program runs.

Python is dynamically typed, meaning that the type of a variable is determined at runtime, and a variable can be reassigned to different types throughout its lifecycle.

In Python, you do not need to declare the type of a variable when you create one. You can change the type of a variable by assigning it a value of a different type.

A screenshot of a code editor window titled 'Basics.py'. The code demonstrates dynamic typing in Python by reassigning a variable 'x' from an integer to a string. The first part shows 'x' being assigned the value 10 and its type being printed as '<class 'int'>'. The second part shows 'x' being reassigned to the string 'Hello' and its type being printed as '<class 'str'>'.

```
# Initially an integer
x = 10
print(type(x))  # <class 'int'>

# Reassigned to string
x = "Hello"
print(type(x))  # <class 'str'>
```



Keywords & Identifiers:

The commanders & names

Keywords are reserved words in Python with special meanings that guide the interpreter on what to do.

They cannot be used as identifiers (e.g., if, else, while, for, def, class, import)

.

There are **33 keywords** in python.

Identifiers are names used to identify variables, functions, classes, modules, or objects.

They start with a letter (**A-Z or a-z**) or an underscore (**_**), followed by letters, underscores, or digits (0-9).



Keywords & Identifiers: The commanders & names

Example :

```
Basics.py

# Keywords in use
if True:
    print("This condition is always true")

# Identifiers: variable, function, and class names
variable_name = 10
def my_function():
    return "Hello, World!"

class MyClass:
    pass

# Output to demonstrate
print(type(variable_name)) # <class 'int'>
print(my_function())       # "Hello, World!"
print(MyClass)             # <class '__main__.MyClass'>
```



Operators: The Mathematicians

```
Basics.py

# Arithmetic Operators
a = 10
b = 5

sum = a + b      # Addition
difference = a - b  # Subtraction
product = a * b   # Multiplication
quotient = a / b  # Division
remainder = a % b # Modulus
power = a ** b    # Exponentiation

# Comparison Operators
is_equal = (a == b)      # Equal to
not_equal = (a != b)     # Not equal to
greater_than = (a > b)   # Greater than
less_than = (a < b)      # Less than
greater_or_equal = (a >= b) # Greater than or equal to
less_or_equal = (a <= b) # Less than or equal to

# Logical Operators
and_operator = (a > 5 and b < 10) # Logical AND
or_operator = (a > 15 or b < 10)  # Logical OR
not_operator = not(a > 5)         # Logical NOT

# Bitwise Operators
bitwise_and = a & b      # Bitwise AND (0)
bitwise_or = a | b       # Bitwise OR (15)
bitwise_xor = a ^ b      # Bitwise XOR (15)
bitwise_not = ~a         # Bitwise NOT (-11)
left_shift = a << 2      # Left Shift (40)
right_shift = a >> 2     # Right Shift (2)

# Membership Operators
my_list = [1, 2, 3, 4, 5]
in_operator = (a in my_list)      # Membership check (False)
not_in_operator = (b not in my_list) # Membership check (False)
```



Conditional Statements: The Decision Makers

Conditional statements allow us to steer the flow of our program, making it dynamic and responsive to different scenarios. Using `if`, `else`, and `elif`, we can execute code based on conditions

.

- **if:** Checks a condition and executes the block if the condition is true.
- **elif:** Checks another condition if the previous conditions were false.
- **else:** Executes the block if none of the above conditions are true.



Conditional Statements: The Decision Makers

Example:

```
Basics.py

# Example of conditional statements
num = 10

if num > 0:
    print("Number is positive")
elif num < 0:
    print("Number is negative")
else:
    print("Number is zero")

# Output
Number is positive
```



Loops : The Repeaters

Loops in Python allow us to execute a block of code repeatedly.

There are two main types of loops:

- **For Loop:** Executes a block of code a specified number of times.
- **While Loop:** Executes a block of code as long as a specified condition is true.

Additionally, loops can be **nested**, meaning one loop inside another.

Real-Life Example: Email Inbox

In your email inbox, you receive multiple messages daily. Using loops, you can automate tasks like sorting, filtering, or marking emails as read/unread. Without loops, you'd need to manually process each email, which would be time-consuming and inefficient. They allow us to automate repetitive tasks efficiently.



Loops : The Repeaters

Example:

```
Basics.py

# Example of for loop
print("Example of for loop:")
for i in range(5):
    print("Iteration", i+1)
# Output
# Iteration 1, Iteration 2, Iteration 3, Iteration 4, Iteration 5

# Example of while loop
print("\nExample of while loop:")
num = 0
while num < 5:
    print("Iteration", num+1)
    num += 1
# Output
# Iteration 1, Iteration 2, Iteration 3, Iteration 4, Iteration 5

# Example of nested loops
print("\nExample of nested loops:")
for i in range(3):
    for j in range(2):
        print("Outer loop:", i, "Inner loop:", j)
# Output
# Outer loop: 0 Inner loop: 0
# Outer loop: 0 Inner loop: 1
# Outer loop: 1 Inner loop: 0
# Outer loop: 1 Inner loop: 1
# Outer loop: 2 Inner loop: 0
# Outer loop: 2 Inner loop: 1
```



Loop control statements: Flow Regulators

In python there are 3 loop control statements:

1. **break Statement:**

- The break statement terminates the loop prematurely when a condition is met.

2. **continue Statement:**

- The continue statement skips the current iteration of the loop when a condition is met and proceeds to the next iteration.

3. **pass Statement:**

- The pass statement is a null operation; nothing happens when it executes. It is used as a placeholder where code is expected but not required.



Loop control statements: Flow Regulators

Example:

```
Basics.py

# Example of break statement
print("Example of break statement:")
for i in range(5):
    if i == 3:
        break
    print("Iteration", i+1)
# Output
# Iteration 1, Iteration 2

# Example of continue statement
print("\nExample of continue statement:")
for i in range(5):
    if i == 2:
        continue
    print("Iteration", i+1)
# Output
# Iteration 1, Iteration 3, Iteration 4, Iteration 5

# Example of pass statement
print("\nExample of pass statement:")
for i in range(5):
    if i == 2:
        pass
    else:
        print("Iteration", i+1)
# Output
# Iteration 1, Iteration 3, Iteration 4, Iteration 5
```



Loop control statements: Flow Regulators

Real-Life Example: Traffic Management System

Consider a traffic management system at a busy intersection. Loop control statements can be likened to the traffic signals and control mechanisms used to regulate the flow of vehicles:

1. **break Statement:**

- Emergency vehicles trigger a green light to bypass traffic.

2. **continue Statement:**

- Pedestrian crossing temporarily pauses vehicle traffic.

3. **pass Statement:**

- System ignores minor issues, like sensor glitches, and continues normal operation.



Loop control statements: Flow Regulators

Example:

```
Basics.py

# Example of loop control statements in a traffic management system
traffic = ["car", "car", "pedestrian", "car", "emergency", "car"]

for vehicle in traffic:
    if vehicle == "emergency":
        print("Green light for emergency vehicle to pass.")
        break # Allow emergency vehicle to pass
    elif vehicle == "pedestrian":
        print("Traffic paused for pedestrian crossing.")
        continue # Allow pedestrians to cross
    else:
        print("Normal traffic flow maintained.")
        pass # Ignore minor issues and continue normal operation

# Output
# Normal traffic flow maintained.
# Normal traffic flow maintained.
# Traffic paused for pedestrian crossing.
# Normal traffic flow maintained.
# Green light for emergency vehicle to pass.
# Normal traffic flow maintained.
```



Bonus Tips:

Print() Function Parameters:

```
# Python's print() Function Parameters:

# sep Parameter:
# Specifies the separator between printed items.
# Default: Space.
# Example:
print("Hello", "World", sep=" ", " ")
# Output: Hello, World

# end Parameter:
# Specifies the character/string printed at the end.
# Default: Newline (\n).
# Example:
print("Hello", end="!!!")
# Output: Hello!!!
```





Ready to Level Up?

CONNECT FOR MORE INSIGHTS AND
FUTURE UPDATES!



Shruti Bharat

@shrutibharat0105