Python Notes

by Gen-Z IITian

About & Quick Links

Gen-Z IITian by Sriram

Topic: Python Programming – Basics to Advanced Notes, Examples, PYQs

YouTube: Gen-Z IITian

Personal YouTube: SriRam_in

Instagram: @curious_sri

Community: IITM BS Unofficial Community

Resources: IIT Pathshala Study Materials

Best Lecture: Foundation Term-1 Course

Reference Links

- **Python Docs:** Official Documentation
- **Practice Problems:** LeetCode
- **IITM BS Resources:** IIT Pathshala
- **Online Editors:** Programiz, Replit, Google Colab
- **Community:** WhatsApp Link

Introduction to Variables

Definition

A **variable** is like a container that stores a value. You can use the variable name in your program instead of writing the value directly. The value of a variable can change as the program runs.

Basic Variable Assignment

Code Example

```
print(10)
```

Output

```
10
```

Now let's assign the value to a variable and print it:

Code Example

```
a = 10 print(a)
```

Output

```
10
```

—

Using Multiple Variables

Code Example

```
a = 10 b = 20 print(a) print(b) print("Sriram is learning with GenZ IITian!")
```

Output

```
10 20 Sriram is learning with GenZ IITian!
```

—

Arithmetic with Variables

Code Example

a = 10 b = 20 print(a + b) addition print(a * b) multiplication

Output

30 200

—

Incrementing a Variable

In programming, the statement `a = a + 1` means "take the current value of `a`, add 1, and store it back into `a`." This is called **incrementing**.

Code Example

a = 10 print(a)
a = a + 1 print(a)
print("Now a is updated! – GenZ IITian")

Output

10 11 Now a is updated! – GenZ IITian

—

Taking Input from User

We can ask the user for input and store it in a variable. `input()` always returns a string, so we convert it to an integer with `int()`.

Code Example

n = int(input("Enter a number, Sriram: ")) print(n) print(n + 1) print(n + 2) print(n + 3)

Sample Output

Enter a number, Sriram: 100
100 101 102 103

Key Takeaways

Summary

- A variable stores a value which can change during execution.
- You can perform arithmetic with variables just like numbers.
- `a = a + 1` is a common way to increment variables.
- Use `input()` (with `int()` for numbers) to accept user input.
- Always personalize and experiment! Example: `print("Hello Sriram from GenZ IITian!")`

Variables and Input Statement

Definition / Idea

A **variable** is a named container that stores a value. The `input()` function pauses the program and reads text typed by the user. Use `int(input(...))` to convert input into an integer; use plain `input(...)` for strings.

Basic: print vs input

Code Example

```
# direct print
print("Hello, type in your name")

# merged: show prompt inside input()
name = input("Hello, type in your name: ")
print("Hello", name)
```

Sample Output

Hello, type in your name: Sriram Hello Sriram

Hinglish Tip

`input("...")` ke andar jo string doge, woh message user ko dikhai dega aur cursor wait karega input ke liye. `input()` hamesha string return karta hai — agar number chahiye to `int()` use karo.

Keep values in separate variables (don't overwrite)

Bad: overwriting the same variable

```
n = input("Type in your name: ")
# oops | reusing n for place overwrites the name
n = input("Which place are you in? ")
print("Hello", n)   # now prints place, not the name
```

Output (shows lost value)

```
Type in your name: Sriram Which place are you in? Mysore
Hello Mysore
```

Fix / Hint

Use different variable names for different inputs (e.g., `name` and `place`) so one value doesn't overwrite another.

Correct: name, place and formatted greeting

Code Example

```
name = input("Type your name: ")
place = input("Which place are you in? ")
# f-string gives clean formatting without extra spaces
print(f"Hello {name}, how is the weather in {place}?")
```

Sample Output

Type your name: Sriram Which place are you in? Mysore
Hello Sriram, how is the weather in Mysore?

Numeric input: convert with `int()`

Code Example

```
age = int(input("What is your age, Sriram? "))
print("Good to know that you are", age, "years old.")
```

Sample Output

What is your age, Sriram? 21
Good to know that you are 21 years old.

Hinglish Note

`int(input(...))` se input string ko integer me convert karte hain. Agar user non-numeric text dega toh `ValueError` aayega — isko hum aage try/except se handle karna seekhenge.

Common error: using an undefined variable

Code (causes NameError)

```
name = input("Type in your name: ")
# forgot to define 'place'
print("Hello", name, "how is the weather in", place)
```

Runtime Error

NameError: name 'place' is not defined

Tip

If you see `NameError: name 'X' is not defined`, it means you tried to use a variable that hasn't been assigned yet (typo or forgot to input/assign).

Mini exercise (try this)

Task

Write a small interactive program that:

1. Asks the user's name and prints a greeting including "GenZ IITian".
2. Asks the user's city and age, then prints:
   `"Hello <name> from <city> | GenZ IITian wishes you a great day! You are <age> years old."`

Hinglish Hint

Use `name = input(...)` and `age = int(input(...))`. Prefer `f-strings` for clean output: `print(f"...name...age...")`

Variables and Literals

Definition / Idea

- A **variable** is a container that can store different values at different times.
- A **literal** is the actual fixed value stored inside a variable (e.g., `40`, `"Sriram"`).
- Variables can appear on both sides of the assignment operator (=), but literals can only be on the right-hand side.

Merging Print and Input

Code Example

```
# Earlier: print + input separately
name = input("Type your name: ")
place = input("Type your location: ")
age = int(input("What is your age? "))

print(f"Hello {name}, how is the weather in {place}?")
print(f"Good to know that you are {age} years old.")
```

Sample Output

Type your name: Sriram Type your location: Chennai What is your age? 21
Hello Sriram, how is the weather in Chennai? Good to know that you are 21 years old.

Changing Variable Values

Code Example

```
name = "Sriram"
print(name)

name = "GenZ IITian"
print(name)

age = 20
print(age)

age = age + 1    # increment
print(age)
```

Output

Sriram GenZ IITian 20 21

Hinglish Note

Dekho — variable ek bucket ki tarah hai, jo alag values rakh sakta hai. Literals jaise `"Sriram"`, `20`, `21` fixed hote hain. Bucket badal sakta hai, paani (literal value) alag ho sakta hai.

Variables vs Literals in Equations

Code Example

```
age = 30
age = age + 1   # works: take current value, add 1, store back

# age = 30 + 1 is valid, but this won't work:
# 30 = 30 + 1   #  invalid
```

Key Difference

- **Variable** can be on both LHS and RHS of assignment. - **Literal** can only be on RHS.

Program: Circle Area Example

Code Example

```
r = int(input("Enter the radius of the circle: "))
area = 3.14 * r * r
print(f"Area of the circle with radius {r} is {area}")
```

Sample Output

Enter the radius of the circle: 5 Area of the circle with radius 5 is 78.5

Concept Check

- Here, `r` and `area` are **variables** (values can change). - `3.14` is a **literal** (fixed value of π). - Run again with $r = 15$, you'll get a new `area`, but `3.14` stays constant.

Summary

Quick Recap

- Variables are like containers — they can hold and change values.
- Literals are the actual fixed values assigned to variables.
- Use variables when values may change (e.g., name, age, radius).
- Use literals when the value is permanent (e.g., `3.14` for π).

1 Data Types (Part 1)

In Python, every value you store in a variable is associated with a specific **data type**. Data types tell the computer how to store and manipulate that data. Python automatically detects the type of data you are storing, whether it is a number, decimal, or text. Let's explore this with examples.

Integer, Float, and String

Definition

Integer (int): Whole numbers without decimals.
Float (float): Numbers with decimal points (fractions).
String (str): Sequence of characters, enclosed in quotes.

Python Code

```
n = 10
r = 6.3
s = "sriram"

print(n)
print(r)
print(s)
```

Output

```
10
6.3
sriram
```

Here, the variable `n` is an integer, `r` is a float, and `s` is a string. Python automatically assigns these types.

Checking Data Types

We can verify the type of any variable using the `type()` function.

Python Code

```
print("n is of type:", type(n))
print("r is of type:", type(r))
print("s is of type:", type(s))
```

Output

```
n is of type: <class 'int'>
r is of type: <class 'float'>
s is of type: <class 'str'>
```

Lists in Python

A list is a collection of values stored in a single variable. Lists are written inside square brackets `[ ]`, with elements separated by commas.

Definition

List (list): An ordered collection of items (integers, floats, strings, or even other lists).

Python Code

```
l = [10, 20, 30, 68, 720, 732, "genz iitian"]

print(l)
print(l[0])    # First element
print(l[1])    # Second element
print(l[6])    # String element
```

Output

```
[10, 20, 30, 68, 720, 732, 'genz iitian']
10
20
genz iitian
```

Type of a List

We can also check the type of the list itself:

Python Code

```
print("l is of type:", type(l))
print("The type of l[2] is:", type(l[2]))
```

Output

```
l is of type: <class 'list'>
The type of l[2] is: <class 'int'>
```

Summary

Key Takeaways

- Integers (`int`) are whole numbers.
- Floats (`float`) are decimal numbers.
- Strings (`str`) are text values.
- Lists (`list`) can store multiple items together.
- Python automatically detects data types.

2 Data Types (Part 2)

In the last lecture, we explored integers, floats, strings, and lists. In this chapter, we will learn about the **Boolean data type** and also see how Python allows us to convert between different data types. This process is called **type conversion** or **type casting**.

Boolean Data Type

Definition

Boolean (bool): A data type that can take only two values: `True` or `False`. Note: The first letter `T` in `True` and `F` in `False` must be capital.

Python Code

```
B1 = True
B2 = False

print(B1)
print(B2)
print("B1 is of type:", type(B1))
print("B2 is of type:", type(B2))
```

Output

```
True
False
B1 is of type: <class 'bool'>
B2 is of type: <class 'bool'>
```

—

Type Conversion: Float and String to Integer

We can explicitly convert values to integers using the `int()` function.

Python Code

```
a = int(5.7)              # Float to int
b = int("10")             # String to int

print("a =", a, "| type:", type(a))
print("b =", b, "| type:", type(b))
```

Output

```
a = 5 | type: <class 'int'>
b = 10 | type: <class 'int'>
```

—

Type Conversion: Integer and String to Float

Python Code

```
x = float(9)              # Integer to float
y = float("5.3")          # String to float

print("x =", x, "| type:", type(x))
print("y =", y, "| type:", type(y))
```

Output

```
x = 9.0 | type: <class 'float'>
y = 5.3 | type: <class 'float'>
```

—

Type Conversion: Integer and Float to String

Python Code

```
s1 = str(9)               # Integer to string
s2 = str(5.3)             # Float to string

print("s1 =", s1, "| type:", type(s1))
print("s2 =", s2, "| type:", type(s2))
```

Output

```
s1 = 9 | type: <class 'str'>
s2 = 5.3 | type: <class 'str'>
```

—

Type Conversion to Boolean

Integers to Boolean: All integers except `0` are considered `True`.

Python Code

```
a = bool(10)
b = bool(0)
c = bool(-10)

print(a, b, c)
```

Output

```
True False True
```

Floats to Boolean: Same rule applies. Only `0.0` is `False`.

Python Code

```
print(bool(3.14))
print(bool(0.0))
print(bool(-7.5))
```

Output

```
True
False
True
```

Strings to Boolean: Any non-empty string is `True`. An empty string is `False`.

Python Code

```
print(bool("sriram"))
print(bool("genz iitian"))
print(bool("0"))          # Still True (because it's a string)
print(bool(""))           # Empty string
```

Output

```
True
True
True
False
```

—

Summary

Key Takeaways

- Booleans have only two values: `True` and `False`.
- Type conversion can be done using functions like `int()`, `float()`, `str()`, and `bool()`.
- Integers: non-zero = `True`, zero = `False`.
- Floats: non-zero = `True`, 0.0 = `False`.
- Strings: non-empty = `True`, empty string = `False`.

Operators and Expressions (Part 1)

Introduction

In this lecture, we will explore **operators** and **expressions** in Python. Operators are symbols like $+$, $-$, $\star$, $/$ which perform operations, while expressions combine variables, literals, and operators to produce results.

Definition: Operators and Expressions

Operator: A symbol that performs an operation (e.g., $+$, $\star$).
Expression: A combination of variables, literals, and operators which evaluates to a value.

Basic Examples

Consider the following Python code:

```
n = 3 + 2
print(n)    # Output: 5

print(3 * 2)      # Output: 6
print(3 * 2.6)    # Output: 7.8
```

Output

5
6
7.8

Working with Variables

```
a = 1
b = 2
n = a + b
print(n)    # Output: 3
```

Output

3

If we use strings instead of numbers:

```
a = "Sriram"
b = "GenZ IITian"
```

```
print(a + b)
```

Output

SriramGenZ IITian

Note

When + is used with strings, Python performs **concatenation** (joining them together). However, multiplication or subtraction with strings throws an error.

Operators with Lists

```
a = [1, 2, 3]
b = [7, 9, 15]
print(a + b)
```

Output

[1, 2, 3, 7, 9, 15]

Here, + combines lists by putting them one after another. This is not a mathematical *union*, but a simple concatenation of lists.

Division and Float Values

```
a = 11
b = 15
n = a / b
print(n)
```

Output

0.733333...

Here, division always returns a floating-point value in Python 3.

Operator Precedence

Now, let us explore a slightly tricky example:

```python
n = 10 + 13 * 2
print(n)
```

Expected vs Actual Result

First Guess (by Sriram): (10 + 13) * 2 = 46
Actual Result: 36

Explanation

Python follows the rule of **operator precedence**. * (multiplication) has higher priority than + (addition).
Thus:

$$10 + 13 \times 2 = 10 + 26 = 36$$

Using Brackets

To enforce the order you want, use brackets:

```python
n = (10 + 13) * 2
print(n)
```

Output

46

Key Takeaway

- Operators like +, −, *, / are executed based on precedence.
- Multiplication and division have higher precedence than addition and subtraction.
- Use brackets `()` to make expressions clear and avoid mistakes.

Conclusion

We learned that Python evaluates arithmetic expressions based on operator precedence. However, using brackets makes our intentions clear. In the next lecture, we will explore more about operators and expressions.

Operators and Expressions (Part 2)

Introduction

In this lecture, we continue our discussion on **operators** in Python. They are divided into three categories:

- Arithmetic Operators
- Relational Operators
- Logical Operators

Definition

Arithmetic Operators: Perform mathematical operations. **Relational Operators:** Compare values and return Boolean results. **Logical Operators:** Combine Boolean expressions.

Arithmetic Operators

The arithmetic operators include: `+`, `-`, `*`, `/`, `//`, `%`, `**`.

```
print(2 + 3)    # Addition
print(9 - 1)    # Subtraction
print(5 * 4)    # Multiplication
print(7 / 3)    # Division
```

Output

```
5
8
20
2.3333...
```

Floor Division

```
print(7 // 3)
```

Output

```
2
```

Floor division discards the decimal part and only returns the integer quotient.

Modulus Operator

```
print(7 % 3)
```

Output

1

The modulus operator gives the remainder of division.

Exponential Operator

```
print(6 ** 2)
```

Output

36

Here, `6 ** 2` means $6^2 = 36$.

Relational Operators

These operators compare values and return Boolean results.

```
print(5 > 10)
print(10 > 5)
```

Output

False
True

Greater than or Equal To

```
print(5 > 5)
print(5 >= 5)
```

Output

False
True

Equal To and Not Equal To

```
print(5 == 50)
print(5 == 5)

print(5 != 50)
print(5 != 5)
```

Output

```
False
True
True
False
```

Note

Relational operators always return `True` or `False`.

Logical Operators

Logical operators work on Boolean values: `and`, `or`, `not`.

AND Operator

```
print(True and True)
print(True and False)
print(False and True)
print(False and False)
```

Output

```
True
False
False
False
```

OR Operator

```
print(True or True)
print(True or False)
print(False or True)
print(False or False)
```

Output

True
True
True
False

NOT Operator

```
print(not True)
print(not False)
```

Output

False
True

Key Idea

- `and` → True only if both are True.
- `or` → True if at least one is True.
- `not` → Inverts the Boolean value.

Conclusion

We explored three important operator categories in Python:

- Arithmetic Operators (mathematical calculations)
- Relational Operators (comparisons → Boolean)
- Logical Operators (combining conditions)

These operators are fundamental in writing decision-making programs. Example:

```
name = "Sriram"
age = 20

if age >= 18 and name == "Sriram":
    print("Welcome GenZ IITian!")
```

Output

Welcome GenZ IITian!

Introduction to Strings in Python

IIT Madras Online BS Degree – Lecture Notes

Declaring Strings

Code Example

```
s = "coffee"
t = "bread"

print(s)
print(t)
print(s + t)    # concatenation
```

Sample Output

coffee
bread
coffeebread

Indexing in Strings

Code Example

```
s = "coffee"
print(s[0])    # 'c'
print(s[1])    # 'o'
print(s[2])    # 'f'
```

Sample Output

c
o
f

String Slicing

Code Example

```
s = "coffee"
print(s[1:3])
print(s[1:5])
print(s[3:5])
```

Sample Output

of
offe
fe

Hinglish Note

`s[start:end]` mein `start` index include hota hai, lekin `end` index exclude hota hai.

Concatenation vs Addition

Code Example

```
s = "0123456789"
a = s[4]    # '4'
b = s[7]    # '7'

print(a + b)
```

Sample Output

47

Code Example

```
a = int(s[4])
b = int(s[7])

print(a + b)
```

Sample Output

11

Practice Example

Code Example

```
s = "0123456789"
a = s[3]      # '3'
b = s[8]      # '8'

n = int(a + b)
print(n)

n = int(a) + int(b)
print(n)
```

Sample Output

38
11

Key Takeaway

Hinglish Note

Moral of the story: Data type decide karta hai ki Python ka operator kaise behave karega.

- Strings ke beech + ⇒ Concatenation
- Numbers ke beech + ⇒ Arithmetic Addition
- Explicit conversion (`int()`, `str()`) zaroori hai for correct behavior

Strings — Replication, Comparison & Indexing

IIT Madras Online Degree — Lecture Notes (Sriram / GenZ IITian style)

String Replication

Code Example

```
s = "good"
print(s * 5)          # replicate the whole string 5
    times

# replicate a single character (first character)
print(s[0] * 5)

# branded example
name = "Sriram"
print(name * 2)       # SriramSriram
```

Sample Output

goodgoodgoodgoodgood
ggggg
SriramSriram

Hinglish Note

`string * n` ko **replication** kehte hain — string ko n baar side-by-side jod deta hai. Example: `"GenZ IITian" * 2` ⇒ `"GenZ IITianGenZ IITian"`.

String Comparison (Lexicographic)

Code Example

```
x = "India"
print(x == "India")     # exact match (case-sensitive)
print(x == "india")     # different case -> False

print("apple" > "one")      # lexicographic comparison
print("4" < "10")           # string comparison (
    lexicographic)

# more examples
print("ab" < "az")
print("abcde" < "abcdef")
```

Sample Output

```
True False
False False
True True
```

Hinglish Note

Strings are compared **character-by-character** (lexicographically) using Unicode code points:

- Comparison is **case-sensitive**: `"India"` `"india"`.
- `"apple" > "one"` compares `'a'` vs `'o'` ($'a' <' o'$) so result is `False`.
- When strings represent numbers (like `"4"` and `"10"`), lexicographic order differs from numeric order. If you want numeric comparison, convert to `int()` first.

Why `"4" < "10"` is `False` (example)

Code Example

```
# '4' vs '10' : compare first characters '4' and '1'
print("4" < "10")          # compares '4' and '1' -> False

# numeric comparison (correct if you mean numbers)
print(int("4") < int("10"))    # True
```

Sample Output

```
False True
```

Hinglish Note

Agar aap numbers compare karna chahte ho to pehle `int(...)` ya `float(...)` karo. String comparison alphabetical (lexicographic) rule follow karta hai.

String Indexing (Positive and Negative)

Code Example

```
s = "Python"

# positive indices (0-based)
print(s[0], s[1], s[2], s[3], s[4], s[5])

# negative indices (from the end)
print(s[-1], s[-2], s[-3], s[-4], s[-5], s[-6])
```

Sample Output

```
P y t h o n
n o h t y P
```

Hinglish Note

Negative indexing se last character ko `s[-1]` se access karte hain, second-last ko `s[-2]` se, and so on. Yeh bahut useful hota hai jab aap string ke end se elements chahte hain.

IndexError and `len()`

Code Example

```
s = "hello"
print(len(s))      # length = 5

# accessing s[5] causes IndexError because valid indices
    are 0..4
print(s[5])        # IndexError: string index out of
   range
```

Sample Output

5

Runtime Error

IndexError: string index out of range

Code Example

```
# correct way to get last character:
print(s[len(s)-1])    # or simply print(s[-1])
```

Sample Output

o

Hinglish Note

`len(s)` se string length milegi. Maximum valid index = `len(s)-1`. Agar aap `len(s)` ko index me use karoge, Python `IndexError` dega.

Quick Exercises

Code Example

```
# 1) Replicate your name 3 times using replication.
# 2) Compare "GenZ" and "genz" and explain the result.
# 3) Extract the last 3 characters of a string using
   negative indexing.
```

Hinglish Note

Try these with the interpreter. Use `int(...)` when comparing numeric strings, and practice slicing to become comfortable with ranges and indices.