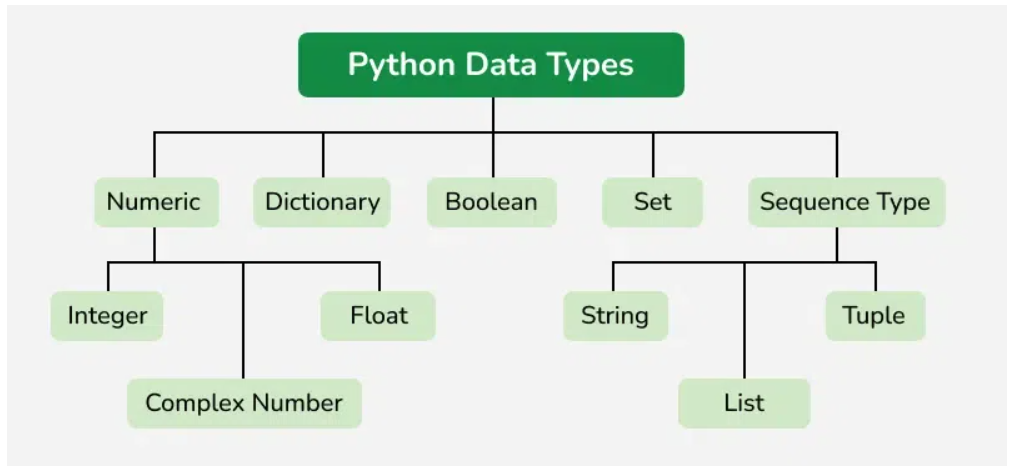
**LECTURE-03 (Sunday 26-April-2025)**

* <https://github.com/panaversity/learn-modern-ai-python/blob/main/00_python_colab/02_data_types/Agentic_AI_Python_Lesson_02_Data_Types.ipynb>
* Every class have its methods and attributes.
* For data, data type selection is very important for operation which we will perform on data.
* Right tool for operation is also important.
* Type Cast is converting one datatype to another datatype.
* Primitive or default datatypes of Python:

Basic datatypes are string and integer while other datatypes are the combination of these two basic datatypes.



* Symbolic AI : human write program and give program to AI machine
* Symbolic AI mostly depends on Boolean datatype.
* SET datatype: It extracts unique elements from repetitive data.
* Numeric Datatypes: Python has three main numeric types:

1. Integer (int) : whole numbers, positive or negative, without decimals.

Example:

num\_int: int = 42

print(type(num\_int)," num\_int = ",num\_int,)

Output:

<class 'int'> num\_int = 42

1. Floating-Point (float) : numbers with decimal points.

Example:

num\_float1: float = 3.14

num\_float2: float = .14

print(type(num\_float1), " num\_float1 = ", num\_float1)

print(type(num\_float2), " num\_float2 = ", num\_float2)

Output:

<class 'float'> num\_float1 = 3.14

<class 'float'> num\_float2 = 0.14

1. Complex (complex): numbers with a real and imaginary part.

Example:

num\_complex: complex = 2 + 3j

print(type(num\_complex), " num\_complex = ", num\_complex)

Output:

<class 'complex'> num\_complex = (2+3j)

Python provides attributes “.real” and “.imag” to extract the real and imaginary parts of a complex number.

Example:

z : complex= 3 + 4j

print("Real Part:", z.real)

print("Imaginary Part:", z.imag)

Output:

Real Part: 3.0

Imaginary Part: 4.0

* Boolean (bool): represents True or False.

Example:

is\_python\_fun: bool = True

is\_python\_difficult : bool = False

print(type(is\_python\_fun), " is\_python\_fun = ", is\_python\_fun)

print(type(is\_python\_difficult), " is\_python\_difficult = ", is\_python\_difficult)

Output:

<class 'bool'> is\_python\_fun = True

<class 'bool'> is\_python\_difficult = False

* Sequence Datatypes: it store multiple items in an ordered way. Below are the four main sequence types.

1. String (str): A sequence of characters enclosed in quotes.

String value can be defined in single or double quotes.

Triple single or double quotes will be used when we have to define value in multi lines.

Example:

text\_double: str  = "Hello, Python!"

text\_single: str  = 'Hello, Python!'

text\_multi: str   = '''Hello, Python!

You are easy

  I will become Python Expert

          Just wait and watch'''

text\_multi\_1: str = """Hello, Python!

You are easy

  I will become Python Expert

          Just wait and watch"""

print(type(text\_double), " text\_double   = ", text\_double)

print(type(text\_single), " text\_single   = ", text\_single)

print(type(text\_multi), " text\_multi    = ", text\_multi)

print(type(text\_multi\_1), " text\_multi\_1  = ", text\_multi\_1)

Output:

<class 'str'> text\_double = Hello, Python!

<class 'str'> text\_single = Hello, Python!

<class 'str'> text\_multi = Hello, Python!

You are easy

I will become Python Expert

Just wait and watch

<class 'str'> text\_multi\_1 = Hello, Python!

You are easy

I will become Python Expert

Just wait and watch

1. List (list): It is an ordered and mutable (changeable after creation) collection.

Example:

my\_list\_1: int = [1, 2, 3, "Java", 3.14, True]

my\_list: list = [1, 2, 3, "Python", 3.14, 3+2j]

print(type(my\_list\_1), " my\_list\_1 = ", my\_list\_1)

print(type(my\_list), " my\_list   =  " + str(my\_list))

Output:

<class 'list'> my\_list\_1 = [1, 2, 3, 'Java', 3.14, True]

<class 'list'> my\_list = [1, 2, 3, 'Python', 3.14, (3+2j)]

Here, “str(my\_list))” is type casting which is converting all elements of “my\_list” into string datatype.

We can set any data type of list like above data type of “my\_list\_1” is “integer” while data type of “my\_list” is “list”.

It is recommended to use same type of elements, which is set for List.

Like above, data type of “my\_list\_1” is “integer” so all elements should be integer type but list items have different datatypes.

Syntax of List: listvariablename : datatypeofelements = [comma sepearted elements]

The position of elements are identified by indexes and index starts with zero.

If we replace square brackets by ( ) then it becomes “Tuple” datatype

If we replace square brackets by { } then it becomes “Set” datatype

Set accepts any type of value separated by comma but dictionary contains key-value pair elements. Curly brackets and colon is shown then dictionary

Example of Dictionary in Python:

my\_dict = {

"name": "Alice",

"age": 30,

"city": "New York"

}

If string value starts with “b” then value is of Byte data type.

Example :

data = b"hello"

data\_item = bytes([104, 101, 108, 108, 111])

print(type(data), " data = ", data)

print(type(data\_item), " data\_item = ", data\_item)

Output:

<class 'bytes'> data = b'hello'

<class 'bytes'> data\_item = b'hello'

* “id” function returns memory address where data is stores in system.

Example:

a : int = 10

b : str = "pqr"

print(id(a))

print(id(b))

Output:

10751144

134718779120304

* Open <https://grok.com/> and use prompt “Give python all data types for student quiz”

Output:

Below is a concise list of all Python data types, tailored for a student quiz, with brief descriptions:

1. **Numeric Types**:
   * **int**: Integer numbers (e.g., 10, -5).
   * **float**: Floating-point numbers with decimals (e.g., 2.5, -0.7).
   * **complex**: Complex numbers with real and imaginary parts (e.g., 1 + 2j).
2. **Sequence Types**:
   * **str**: String, a sequence of characters (e.g., "hello", 'world').
   * **list**: Ordered, mutable collection (e.g., [1, "a", 3.14]).
   * **tuple**: Ordered, immutable collection (e.g., (1, 2, "b")).
3. **Mapping Type**:
   * **dict**: Collection of key-value pairs (e.g., {"name": "Bob", "age": 15}).
4. **Set Types**:
   * **set**: Unordered collection of unique items (e.g., {1, 2, 3}).
   * **frozenset**: Immutable set (e.g., frozenset([1, 2, 3])).
5. **Boolean Type**:
   * **bool**: Logical values True or False.
6. **None Type**:
   * **NoneType**: Represents a null value (None).
7. **Binary Types**:
   * **bytes**: Immutable sequence of bytes (e.g., b"data").
   * **bytearray**: Mutable sequence of bytes (e.g., bytearray(b"data")).
   * **memoryview**: Memory view of an object’s bytes (e.g., memoryview(b"data")).

**Quiz Notes**:

* **Mutability**: Lists, dictionaries, sets, and bytearrays are mutable; tuples, strings, frozensets, and bytes are immutable.
* **Type Checking**: Use type() to identify a data type (e.g., type("hello") returns <class 'str'>).
* **Common Operations**: Know basic operations like indexing for sequences, adding/removing This script can be run to show output and types, making it ideal for a quiz to test understanding of Python data types. Let me know if you need a specific quiz format or additional details!

Output end

* Example of Type Cast : str(123) , here integer value 123 is converted to string value “123”
* <https://github.com/panaversity/learn-modern-ai-python/blob/main/00_python_colab/03_operators_keywords_variables/Agentic_AI_Python_Lesson_03_Operators%2C_Keywords_%26_Variables.ipynb>
* PEP8 naming convention : <https://peps.python.org/pep-0008/#naming-conventions>
* Variable name convention rules:
  1. special character/space not allowed
  2. For variable and function name, use snake case.

Example:

father name : str = “abc” – wrong

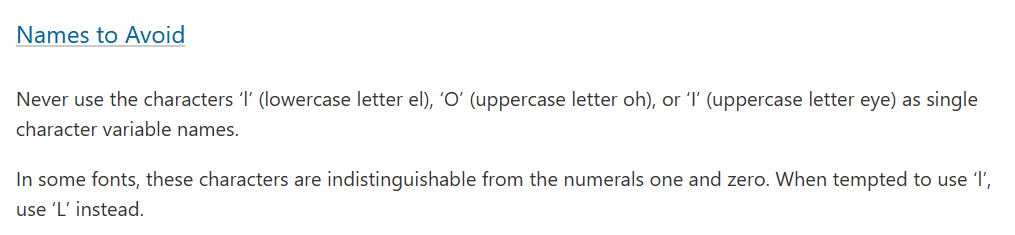
father-name : str = “abc” – wrong

1father\_name : str = “abc” – wrong

father\_name : str = “abc” – correct snake\_case

Name1 : str = “abc” – correct

* 1. Do not use capital letters for variable name



* It is best practice to define static type variable

Example:

Static Type Variable

a : str = "PQR"

Dynamic Type Variable

a = "PQR"

* “fatherName”, it is Camel case naming convention
* Only write variable name and on execution, variable value will be printed

Example:

abc : str = "PQR"

abc

Ouput:

PQR

But in this case, value of variable is printed on execution only when variable name is written in last line and there is no any other line after variable name

Example 1:

abc : str = "PQR"

abc

ghi : str = "XYZ"

ghi

Ouput:

XYZ

Here, “abc” variable value not printed only “ghi” value printed because “ghi” variable is written in last line

Example 2:

abc : str = "PQR"

abc

ghi : str = "XYZ"

ghi

chr : str = "QWE"

On execution no output

* “\n” is used to add newline in string value.

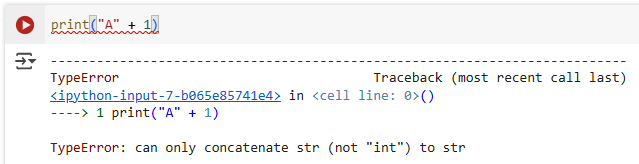


* Concatenation: Combining values or value of string variables. All things which are dynamic in applications is due to concatenation. There are four ways of string concatenation:

1. +

Both sides value should be string.

Wrong



Correct



Correct after type casting



Example 1:

name : str = "Muhammad Ahsan"

father\_name : str = "Muhammad Aslam"

university : str = "Bahria University"

card : str = "PIAIC Student Card\n\nStudent Name : " + name

print(card)

Output:

PIAIC Student Card

Student Name : Muhammad Ahsan

Also if only write variable name in last,

name : str = "Muhammad Ahsan"

father\_name : str = "Muhammad Aslam"

university : str = "Bahria University"

card : str = "PIAIC Student Card\n\nStudent Name : " + name

card

Output:

PIAIC Student Card\n\nStudent Name : Muhammad Ahsan

1. %

Without concatenation, static:

Example:

card : str = "PIAIC Student Card\n\nStudent Name : ABC\nFather Name : DEF\nUniversity : Bahria"

print(card)

Output:

PIAIC Student Card

Student Name : ABC

Father Name : DEF

University : Bahria

Use placeholder, only for values which can be changed, in above static example, university will always be same, so no need place holder for university name.

Example:

name : str = "Muhammad Ahsan"

father\_name : str = "Muhammad Aslam"

university : str = "Bahria University"

age : int = 20

card : str = "PIAIC Student Card\n\nStudent Name : %s\nFather Name : %s\nAge : %d\nUniversity : %s" % (name, father\_name, age, university)

print(card)

Output:

PIAIC Student Card

Student Name : Muhammad Ahsan

Father Name : Muhammad Aslam

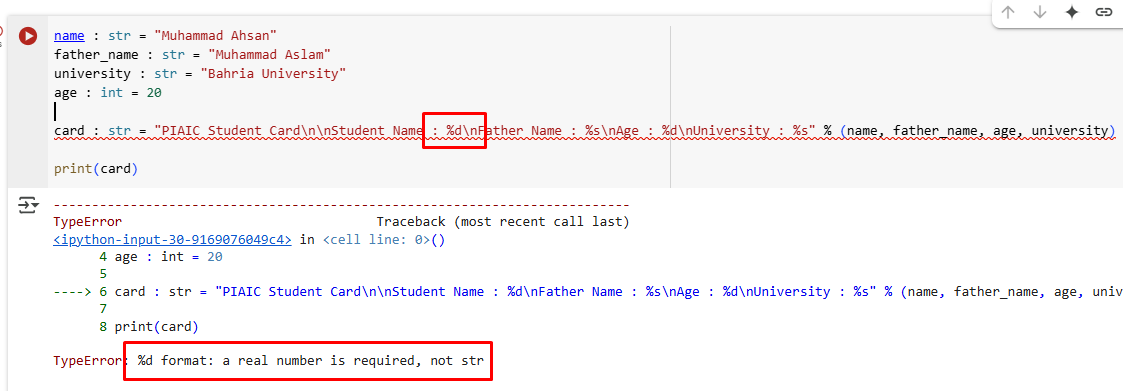
Age : 20

University : Bahria University

But while using “%” for concatenation, we have to make sure that we use proper sign with % inside string value.

%s – string value

If we use any other sign for string value then error raised.

Example: 

1. .format (dot format)

Example:

name : str = "Muhammad Ahsan"

father\_name : str = "Muhammad Aslam"

age : int = 22

university : str = "Bahria University"

card : str = """PIAIC Student Card

Student Name : {}

Father Name : {}

Age : {}

University : {}""".format(name, father\_name, age, university)

print(card)

Output:

PIAIC Student Card

Student Name : Muhammad Ahsan

Father Name : Muhammad Aslam

Age : 22

University : Bahria University

If we by mistake pass invalid variable in sequence then error will not be raised but major issue occurred.

Example:

name : str = "Muhammad Ahsan"

father\_name : str = "Muhammad Aslam"

age : int = 22

university : str = "Bahria University"

card : str = """PIAIC Student Card

Student Name : {}

Father Name : {}

Age : {}

University : {}""".format(name, age, father\_name, university)

print(card)

Output:

PIAIC Student Card

Student Name : Muhammad Ahsan

Father Name : 22

Age : Muhammad Aslam

University : Bahria University

1. f string

Example:

name : str = "Muhammad Ahsan"

father\_name : str = "Muhammad Aslam"

age : int = 22

university : str = "Bahria University"

card : str = f"""PIAIC Student Card

Student Name : {name}

Father Name : {father\_name}

Age : {age}

University : {university}"""

print(card)

Output:

PIAIC Student Card

Student Name : Muhammad Ahsan

Father Name : Muhammad Aslam

Age : 22

University : Bahria University

Class Assignment: use above code and add more variables

from datetime import datetime, date

name : str = "ABC"

father\_name : str = "DEF"

age : int = "10"

is\_male : bool = True

date\_of\_birth : datetime = datetime(2020, 11, 25)

sibilings : list = ["PQR", "STU", "UVW"]

card : str = f"""Student Name : {name}

Father Name : {father\_name}

Age : {age}

Gender : { "Male" if is\_male else "Female"}

Date of Birth : {date\_of\_birth.date()}

Siblings : {sibilings}"""

print(card)

Output:

Student Name : ABC

Father Name : DEF

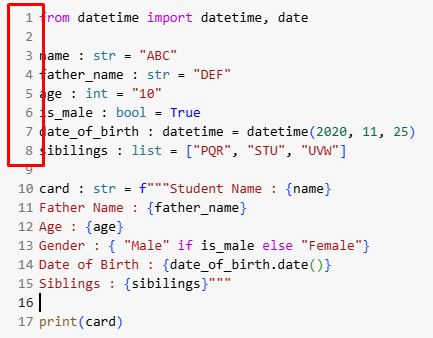
Age : 10

Gender : Male

Date of Birth : 2020-11-25

Siblings : ['PQR', 'STU', 'UVW']

* <https://github.com/adam-p/markdown-here/wiki/markdown-cheatsheet> : it contains information related to tags which we have to add while working on markdown file.
* Ctrl + M + L -> is used to get line numbers



* Python is case sensitive language.

Wrong

father\_name = "PQR"

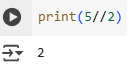
print(father\_Name)

We have to use “father\_name” variable name throughout the application

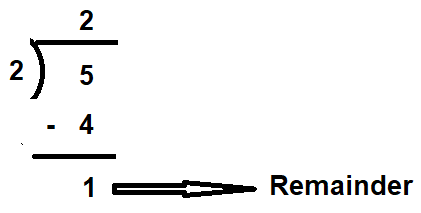
* <https://github.com/panaversity/learn-modern-ai-python/blob/main/00_python_colab/03_operators_keywords_variables/Agentic_AI_Python_Lesson_03_Operators%2C_Keywords_%26_Variables.ipynb>
* Division operator:

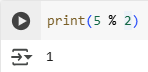


* Floor division: Use it when you do not want division output in decimal value



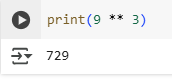
* Modulus operator: used to get remainder.



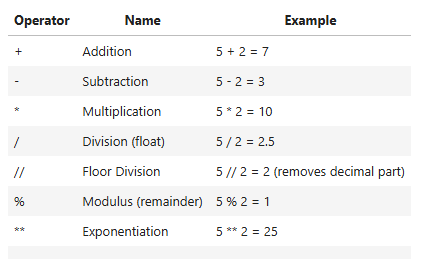


* Exponential





* Arithmetic Operators: used for basic mathematical operations.



Example:

a: int = 10

b: int = 3

print("a + b  = ", a + b)   # 13 Addition

print("a - b  = ", a - b)   # 7 Subtraction

print("a \* b  = ", a \* b)   # 30 Multiplication

print("a / b  = ", a / b)   # 3.3333333333333335

print("a // b = ", a // b)  # 3 Floor Division

print("a % b  = ", a % b)   # 1 Modulus (remainder)

print("a \*\* b  = ", a \*\* b)  # 1000 Exponentiation (10 \* 10 \* 10)

Output:

a + b = 13

a - b = 7

a \* b = 30

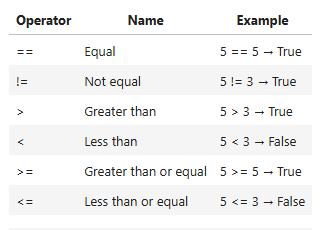
a / b = 3.3333333333333335

a // b = 3

a % b = 1

a \*\* b = 1000

* Comparison (Relational) Operators: used to compare two values.



Example:

x: int = 10

y: int = 5

print("x == y = ", x == y)  # False, Equal

print("x != y = ", x != y)  # True, Not equal

print("x > y  = ", x > y)   # True, Greater than

print("x < y  = ", x < y)   # False, Less than

print("x >= y = ", x >= y)  # True, Greater than or equal

print("x <= y = ", x <= y)  # False, Less than or equal

Output:

x == y = False

x != y = True

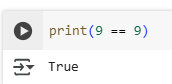
x > y = True

x < y = False

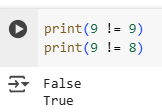
x >= y = True

x <= y = False

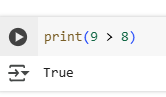
* Equal “==”



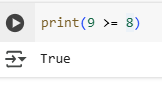
* Not equal to “!=”



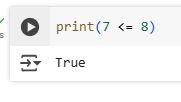
* Greater



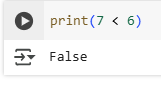
* Greater or equal to



* Less than or equal to



* Less than <



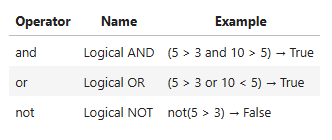
* Equal to === (strict comparison): both side data type is also compared together with values.

9 === “9” -> False

“9” === “9” -> True

Python does not support ===

* Logical operators: used to combine conditional statements.



Example:

x: bool = True

y: bool = False

print("x and y = ", x and y)

print("x or y  = ", x or y)

print("not x   = ", not x)

Output:

x and y = False

x or y = True

not x = False

* and operator: should be or mandatory means all conditions should be true then final output will be true.

Example:

print(True and True and True and True)

print(True and True and True and False)

print(True and True and False and False)

print(True and False and False and False)

print(False and False and False and False)

Output:

True

False

False

False

False

* or operator: maybe or optional means any one condition is true then final output will be true.

Example:

print(True or True or True or True)

print(True or True or True or False)

print(True or True or False or False)

print(True or False or False or False)

print(False or False or False or False)

Output:

True

True

True

True

False

* Example “or” and “and”:

num1 = 58

num2 = 99

num3 = 54

num4 = num1 < num2

num5 = num2 <num3

print(num4)

print(num5)

#OR

print(num2 > num1 or num2 < num3)

#AND

print(num2 > num1 and num2 < num3)

Output:

True

False

True

False

* Example: Check provided number is “Even” or “Odd”

num = int(input("Enter a number:"))

if(num % 2 == 0):

  print("even number")

else:

  print("odd number")

Output:



pass value and you get information if provided number is odd or even.

Ouput:

Enter a number:10

even number

OR

Enter a number:3

odd number

* NOT operator: reverse logical condition

Example 1:

not True

Output:

False

Example 2:

not False

Output:

True

Example 3:

username = input("Enter your username:")

if not username == "Qasim":

  print("username is not Qasim")

else:

  print("username is Qasim")

Output:

Enter your username:Qasim

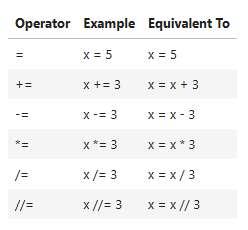
username is Qasim

OR

Enter your username:Abid

username is not Qasim

* Assignment operators: used to assign values to variables.



Example 1:

number : int = 99

number = number + 2

number += 2

number

Output:

103

Here, value overwritten

Example 2:

number : int = 99

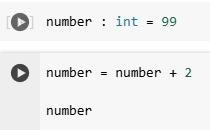
number = number

number

Output:

99

Example 3:



Whenever we only run second code cell then number increased by 2.

Execute Second Cell 1 time : number = 101

Execute Second Cell 1 time : number = 103

Execute Second Cell 1 time : number = 105

But if we run first cell then number = 99 again.

* // operator (floor): used to round value and remove decimal point.

import math

print(5 / 2)

print(5 // 2)

print(math.floor(2.9))

print(math.ceil(2.1))

Output:

2.5

2

2

3

Here, “//” operator, “ceil” method and “floor” method is used to round-off values.

* \*\* (Power) operator



Example:

print(2\*\*2)

print(2\*\*3)

print(2\*\*4)

Output:

4

8

16