Computational Thinking

Lecture 03: Expression, Operators & I/O

University of Engineering and Technology

VIETNAM NATIONAL UNIVERSITY HANOI

Outline

- Real-life Example
- Variable & Type
- Expressions & Operators
- Input & Output in Python

Real-life Example

Real-life Example: Bookstore Cashier (1)

Task: Write a program to support bookstore cashiers to calculate the total payment.

Example:

#	Name	Qty	Price ($)	Ammount ($)
1	Book A	1	10.0	10.0
2	Book B	2	2.5	5.0
			Total:	15.0

Real-life Example: Bookstore Cashier (2)

Task: Write a program to support bookstore cashiers to calculate the total payment.

- Input: List of items (book) in a given basket.
- Decomposition:
  - Total \$ = ∑ \$ of items in the bill
- Pattern Recognition & Abstraction:
  - \$ amount (each item) = qty x price
- Algorithm Design:
  - Initialize the total payment = 0
  - Consider each item in the bill
    - Increase the total payment of \$ amount
  - Print total

Real-life Example: Bookstore Cashier (3)

Task: Write a program to support bookstore cashiers to calculate the total payment.

Variable	Data Type	Value
Item_name	str (text)	Book A
Qty	int	2
Price	float	2.5

- Data
  - Item Name: Text
  - Qty: Integer Value
  - Price/Amount/Total: Real Value
- Caculation:

$$\underbrace{\text{amount = qty x price}}_{\text{Expression}}$$

Operator

Variable, Data Type & Value, Operator

$$\underbrace{\text{amount = qty x price}}_{\text{Expression}}$$

Operator

Variable	Data Type	Value
Item_name	str (text)	Book A
Qty	int	2
Price	float	2.5

A variable's data type defines its value type and the operations (operators) that can perform on it.

Note (*): The data type can change during program execution

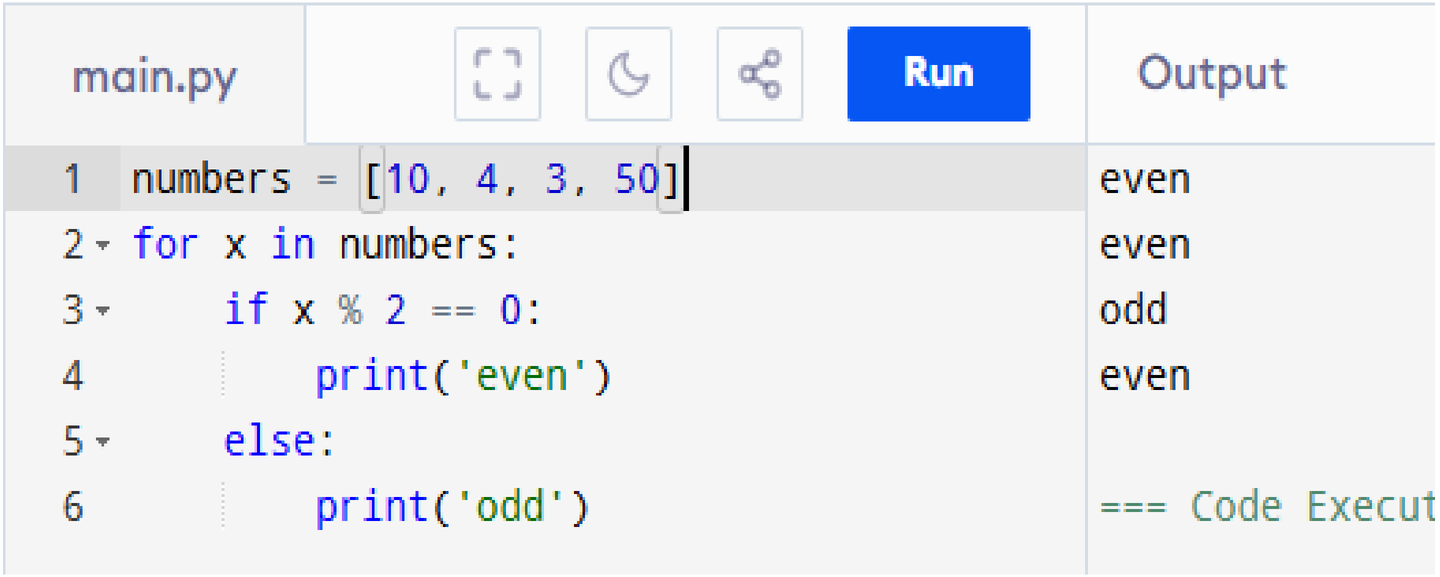
Data Type

- Numeric: 10, -2.5
- Boolean: True False
- Set: (‘Math’, 'Engligh’, ‘Physics’)
- Dictionary: {'Math': 8.4, 'English': 9.0, 'Physics': 6.5}
- Sequence:
  - Strings 'Hello' "Hello" "It's a good day, today"
  - List [1,2,3] ["it's", "a", "good", "day"]
  - Tuple ('Math', 8.4) (‘John', ‘English', 84)
- None

Variable, Type & Variable Assignment

Programming with Python

Given a set of instructions and a task, write a sequence of instructions that do the task.

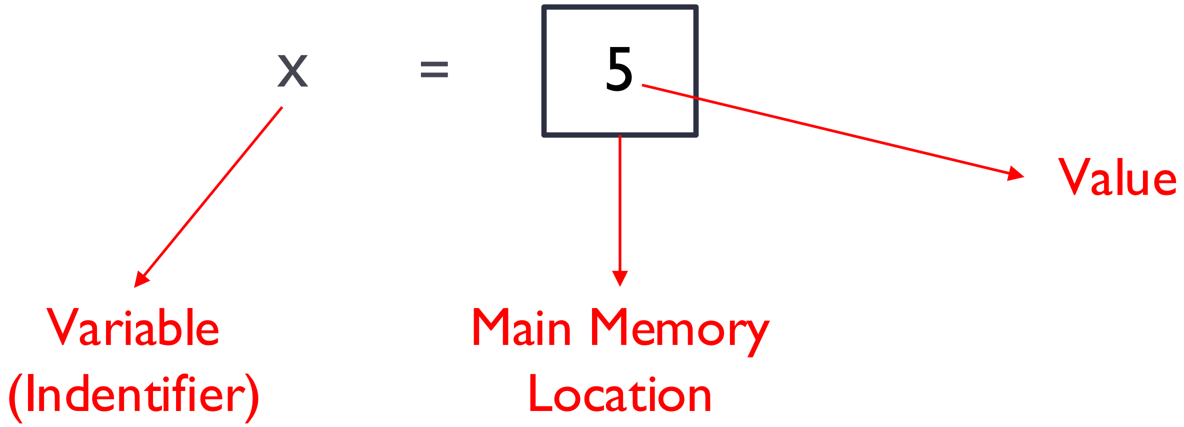


```python
numbers = [10, 4, 3, 50]
for x in numbers:
    if x % 2 == 0:
        print('even')
    else:
        print('odd')
```

```
even
even
odd
even

=== Code Execut
```

Variables

Variables are used to store data that can be referenced and manipulated during program execution.



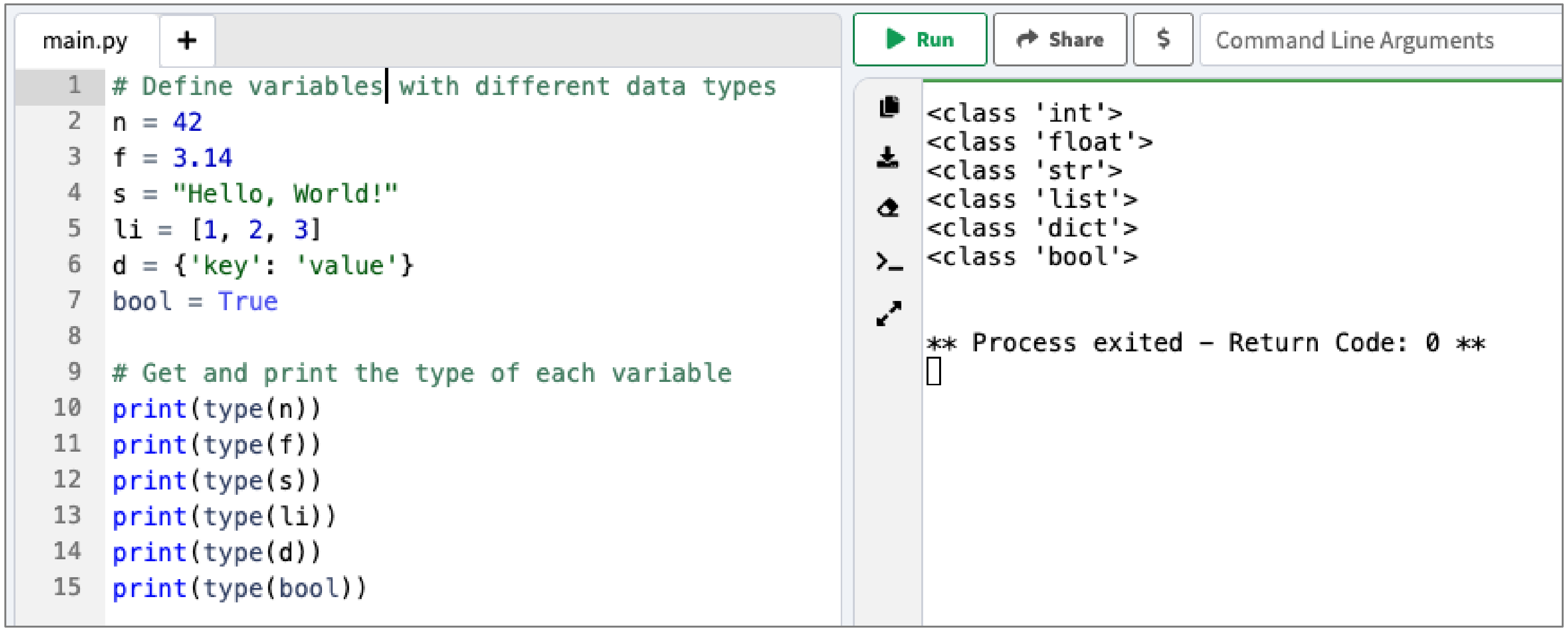
A variable is essentially a name that is assigned to a value

Rules for Naming Variables in Python

- Variable names can only contain letters, digits and underscores (_), e.g., total_payment, _var
- A variable name cannot start with a digit, e.g., ~~2var_name~~
- Variable names are case-sensitive, e.g., myVar ≠ myvar.
- Avoid using Python keywords, e.g., *if, else, for* as variable names.

Type of Variable

In Python, we can determine the type of a variable using the *type()* function



```python
# Define variables with different data types
n = 42
f = 3.14
s = "Hello, World!"
li = [1, 2, 3]
d = {'key': 'value'}
bool = True

# Get and print the type of each variable
print(type(n))
print(type(f))
print(type(s))
print(type(li))
print(type(d))
print(type(bool))
```

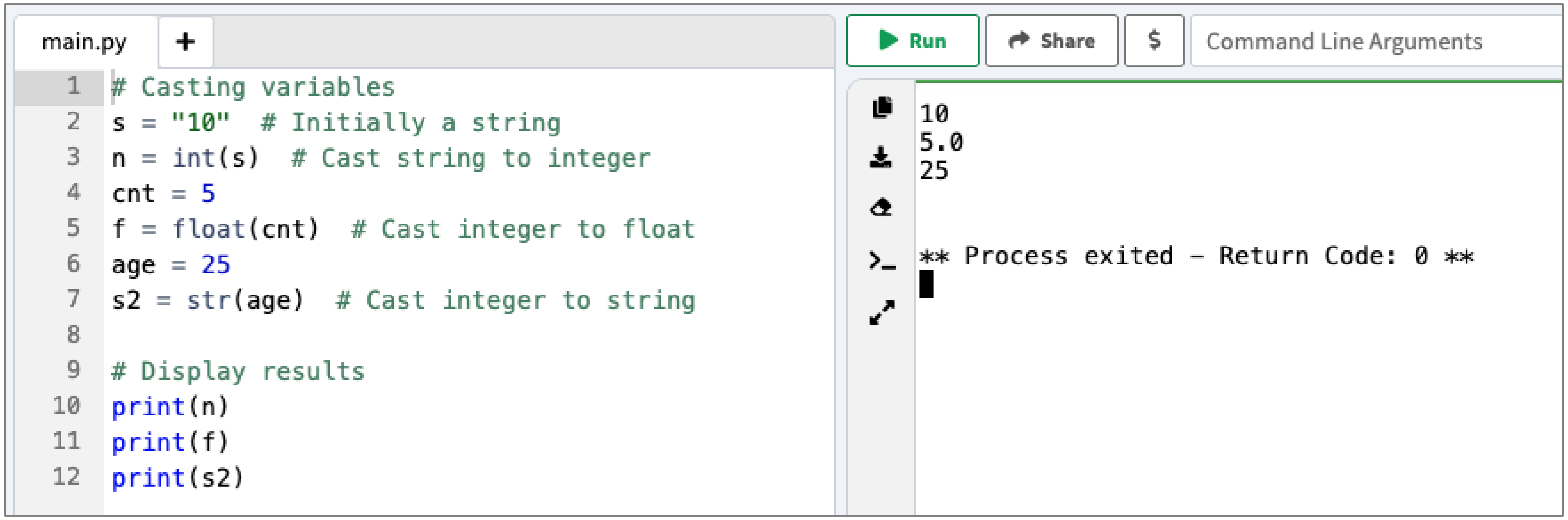
```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'list'>
<class 'dict'>
<class 'bool'>


** Process exited - Return Code: 0 **
```

Assigning Values to Variables

- Basic Assignment
  - x = 5
  - y = 3.14
  - s = “Hello”
- Dynamic Typing
  - x = 10
  - x = “Now a string”
- Multiple Assignments
  - Same value: a = b = c = 5
  - Different values: x, y, z = 1, 2.5, “Python”

Type Casting a Variable

Type casting refers to the process of converting the value of one data type into another.



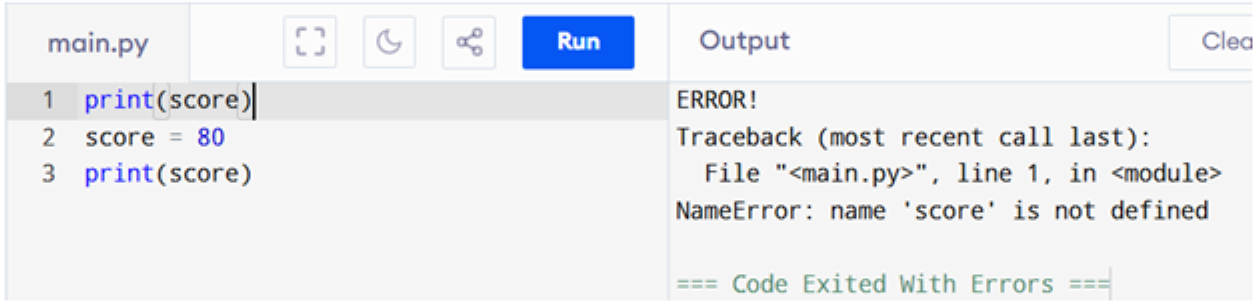
```python
# Casting variables
s = "10"  # Initially a string
n = int(s)  # Cast string to integer
cnt = 5
f = float(cnt)  # Cast integer to float
age = 25
s2 = str(age)  # Cast integer to string

# Display results
print(n)
print(f)
print(s2)
```

```
10
5.0
25


** Process exited - Return Code: 0 **
```

Variables…

We need to define a name before it can be used

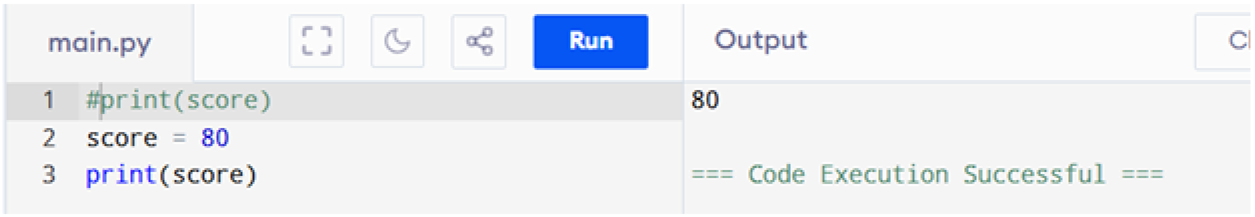
score is used before being defined



```
print(score)
score = 80
print(score)
```

```
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 1, in <module>
NameError: name 'score' is not defined

=== Code Exited With Errors ===
```

score is used after having been defined

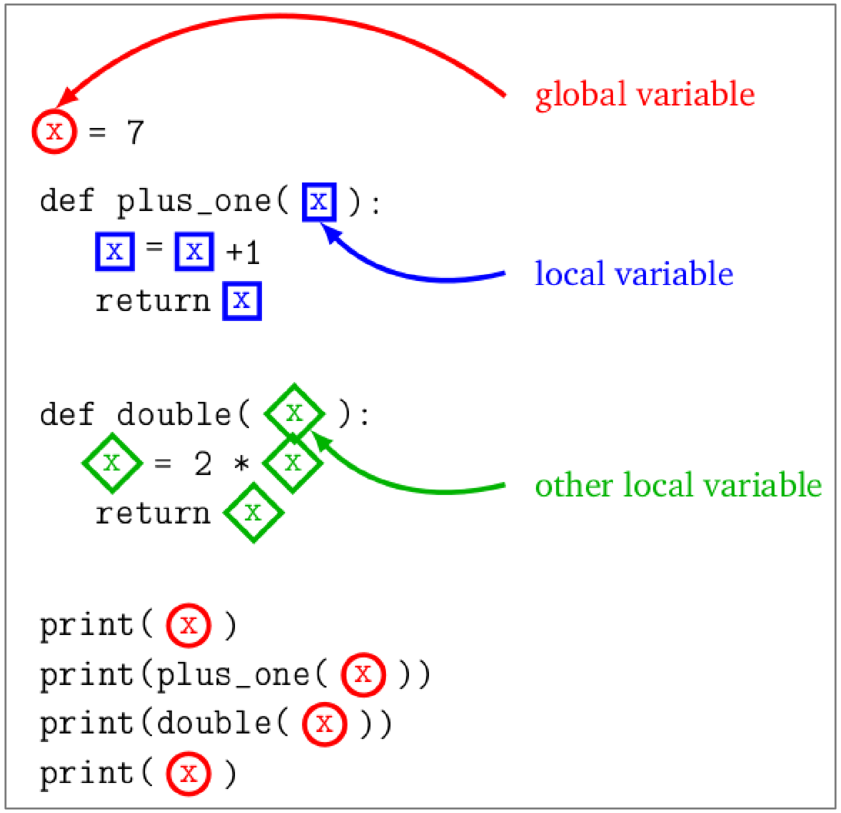


```
#print(score)
score = 80
print(score)
```

```
80

=== Code Execution Successful ===
```

Local & Global Variables

- Local variables:
  - Declare inside a function, exist only during its execution.
  - Cannot be accessed from outside the function.
- Global variables
  - Declare outside all functions
  - Can be accessed anywhere in the program, including inside functions.



```
x = 7

def plus_one( x ):
    x = x +1
    return x

def double( x ):
    x = 2 * x
    return x

print( x )
print(plus_one( x ))
print(double( x ))
print( x )
```

Expression & Operators

Expression

An expression is a combination of operators and operands that is interpreted to produce some other value.

- Example:

$$\underbrace{\texttt{x = 3 + 5}}_{\text{Simple Expression}}$$

$$\underbrace{\texttt{x = (3 * 7 + 2) * 0.1}}_{\text{Complex Expression}}$$

Python Operators

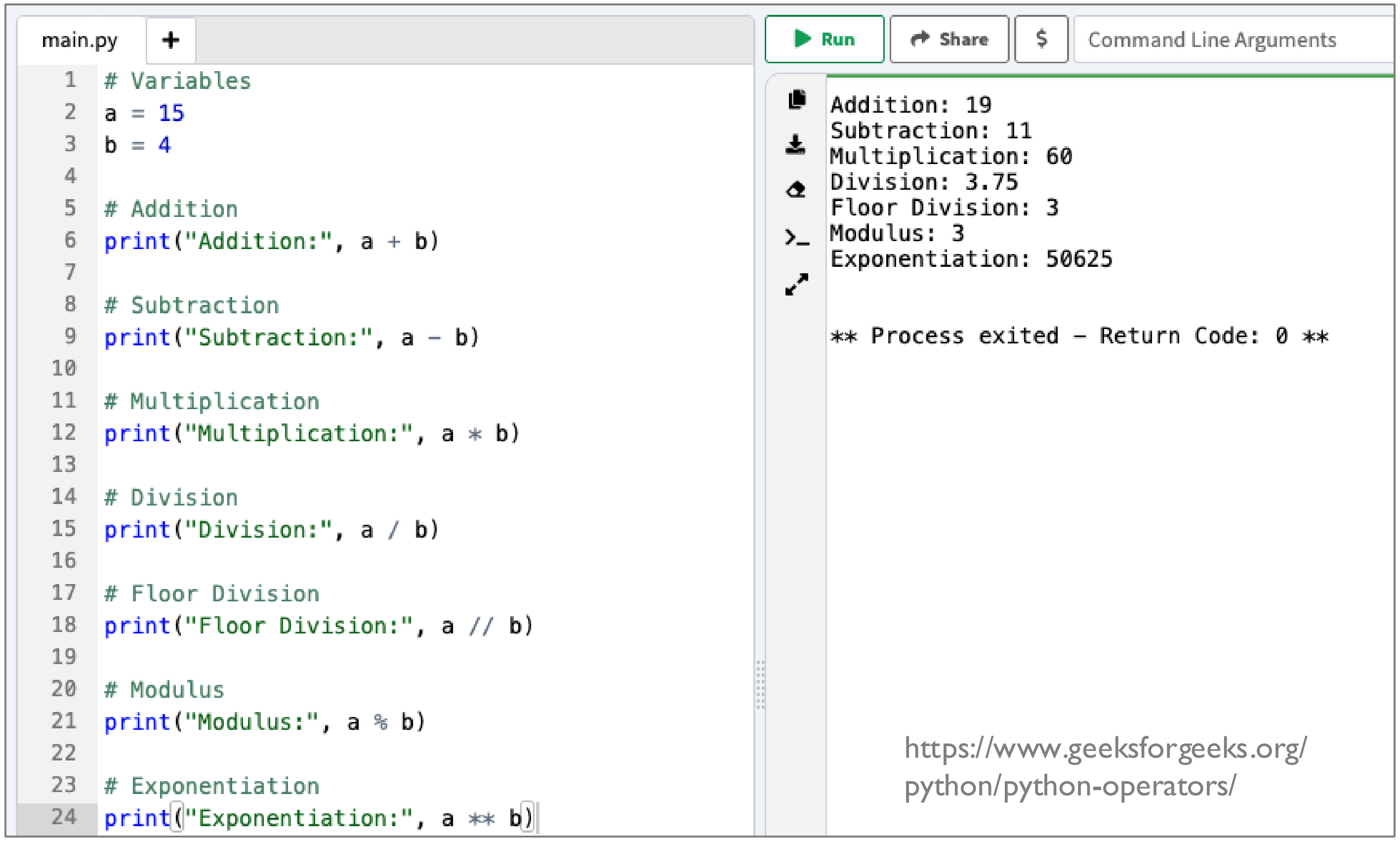
Operators in general are used to perform operations on values and variables

#	Type	Operators
1	Arithmetic operator	+, -, *, /, %
2	Comparison operator	<, <=, >, >=, ==, !=
3	Logical operator	AND, OR, NOT
4	Bitwise operator	&, |, <<, >>, ~, ^
5	Assignment operator	=, +=, -=, *=, %=
6	Membership Operators	in, not in

https://www.geeksforgeeks.org/python/python-operators/

Arithmetic Operators

Use to perform basic mathematical operations like addition, subtraction, multiplication and division.

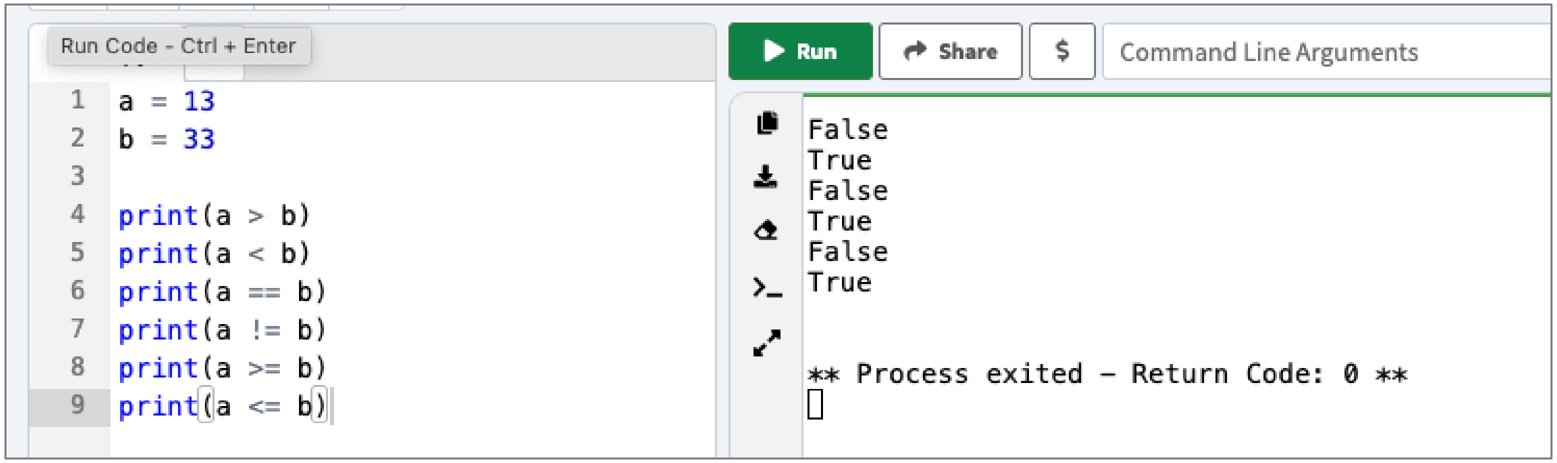


```python
# Variables
a = 15
b = 4

# Addition
print("Addition:", a + b)

# Subtraction
print("Subtraction:", a - b)

# Multiplication
print("Multiplication:", a * b)

# Division
print("Division:", a / b)

# Floor Division
print("Floor Division:", a // b)

# Modulus
print("Modulus:", a % b)

# Exponentiation
print("Exponentiation:", a ** b)
```

```
Addition: 19
Subtraction: 11
Multiplication: 60
Division: 3.75
Floor Division: 3
Modulus: 3
Exponentiation: 50625


** Process exited - Return Code: 0 **
```

https://www.geeksforgeeks.org/python/python-operators/

Comparison Operators



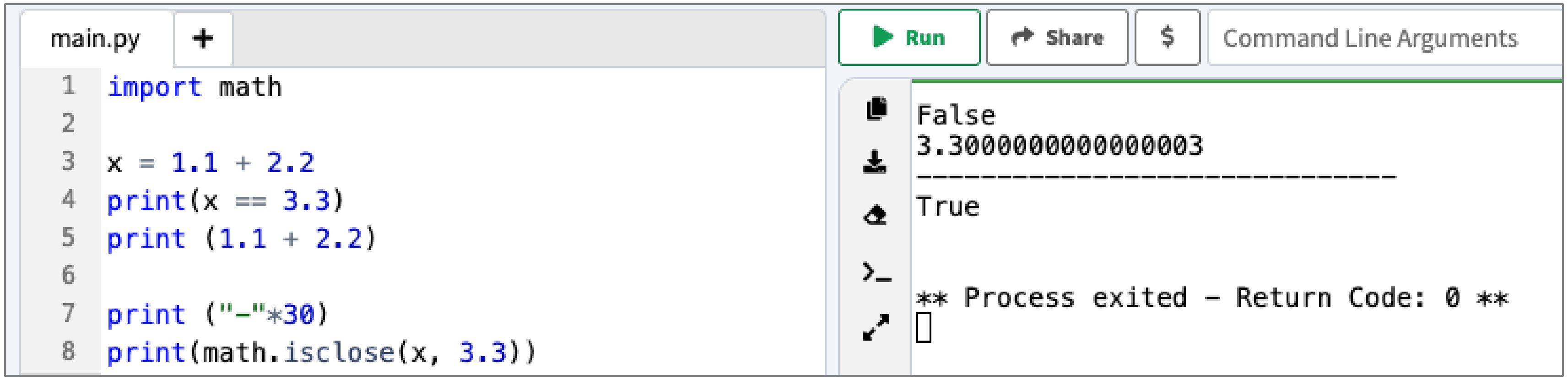
```python
a = 13
b = 33

print(a > b)
print(a < b)
print(a == b)
print(a != b)
print(a >= b)
print(a <= b)
```

```
False
True
False
True
False
True


** Process exited - Return Code: 0 **
```

Comparison (or Relational) operators compares values. It either returns True or False according to the condition.

https://www.geeksforgeeks.org/python/python-operators/

Comparison of Floating-Point Values



```python
import math

x = 1.1 + 2.2
print(x == 3.3)
print (1.1 + 2.2)

print ("-"*30)
print(math.isclose(x, 3.3))
```

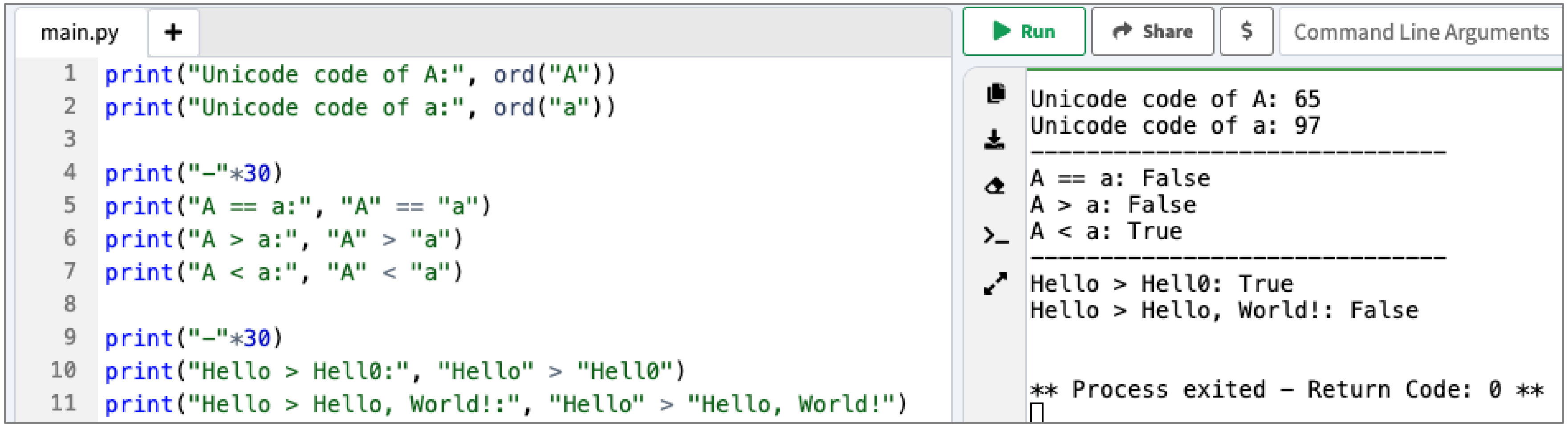
```
False
3.3000000000000003
------------------------------
True


** Process exited - Return Code: 0 **
```

It is bad practice to compare floating-point values for exact equality using the == operator.

- Solution: math.isclose()

https://realpython.com/python-operators-expressions/

Comparison of Strings



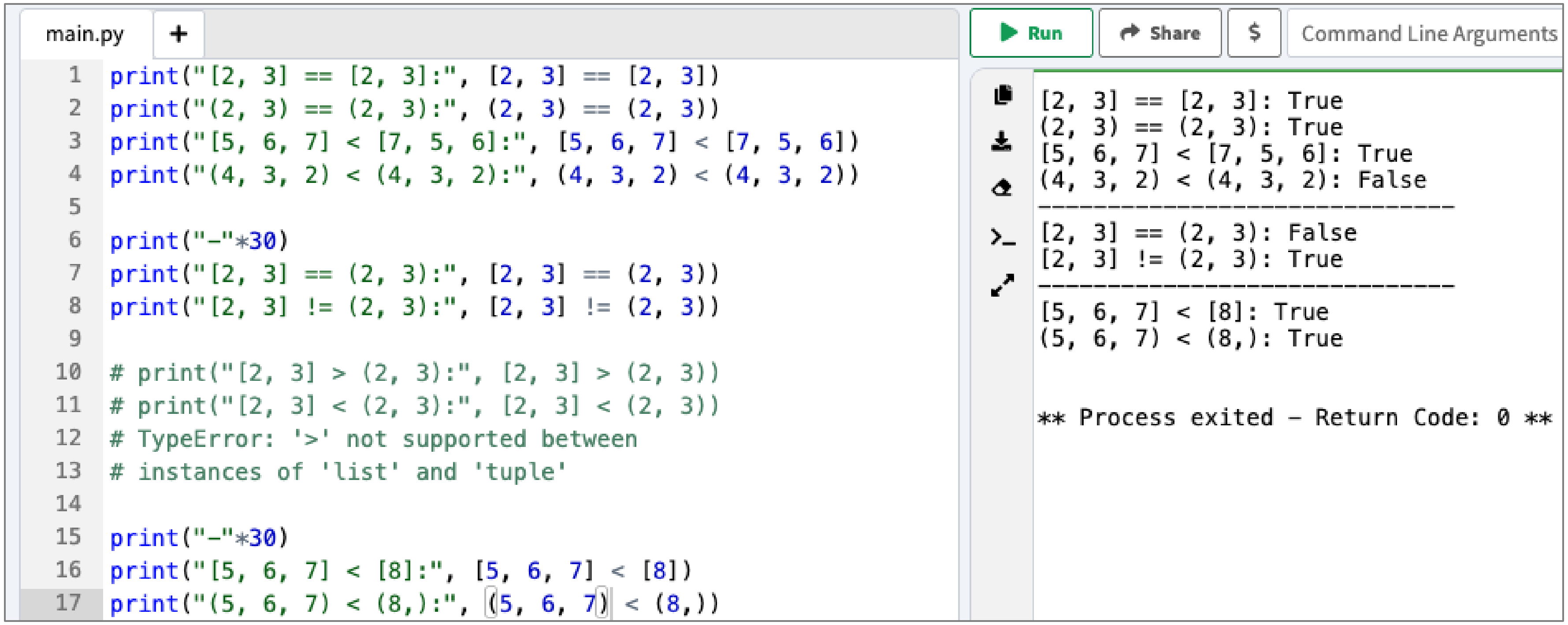
```python
print("Unicode code of A:", ord("A"))
print("Unicode code of a:", ord("a"))

print("-"*30)
print("A == a:", "A" == "a")
print("A > a:", "A" > "a")
print("A < a:", "A" < "a")

print("-"*30)
print("Hello > Hell0:", "Hello" > "Hell0")
print("Hello > Hello, World!:", "Hello" > "Hello, World!")
```

```
Unicode code of A: 65
Unicode code of a: 97
------------------------------
A == a: False
A > a: False
A < a: True
------------------------------
Hello > Hell0: True
Hello > Hello, World!: False


** Process exited - Return Code: 0 **
```

Python compares strings character by character using each character's Unicode code point.

https://realpython.com/python-operators-expressions/

Comparison of Lists, Tuples



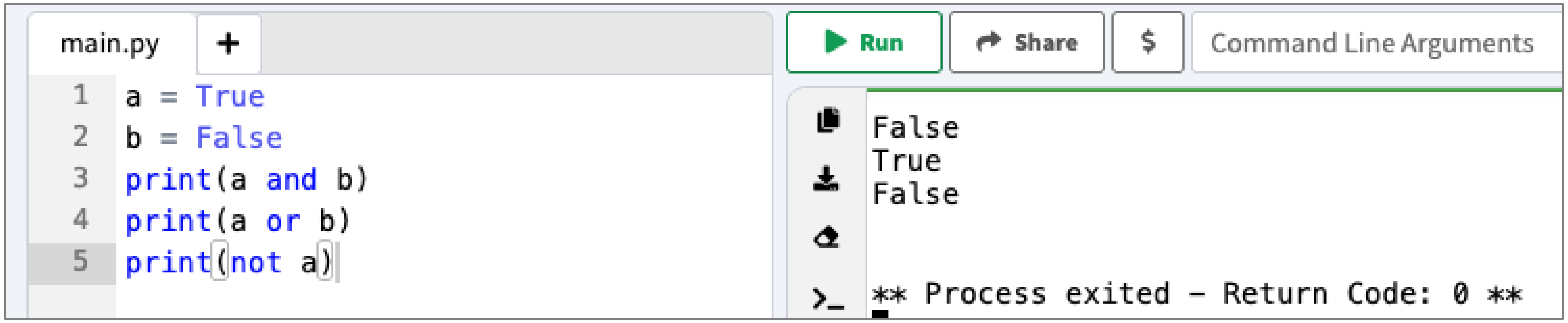
```
print("[2, 3] == [2, 3]:", [2, 3] == [2, 3])
print("(2, 3) == (2, 3):", (2, 3) == (2, 3))
print("[5, 6, 7] < [7, 5, 6]:", [5, 6, 7] < [7, 5, 6])
print("(4, 3, 2) < (4, 3, 2):", (4, 3, 2) < (4, 3, 2))

print("-"*30)
print("[2, 3] == (2, 3):", [2, 3] == (2, 3))
print("[2, 3] != (2, 3):", [2, 3] != (2, 3))

# print("[2, 3] > (2, 3):", [2, 3] > (2, 3))
# print("[2, 3] < (2, 3):", [2, 3] < (2, 3))
# TypeError: '>' not supported between
# instances of 'list' and 'tuple'

print("-"*30)
print("[5, 6, 7] < [8]:", [5, 6, 7] < [8])
print("(5, 6, 7) < (8,):", (5, 6, 7) < (8,))
```

```
[2, 3] == [2, 3]: True
(2, 3) == (2, 3): True
[5, 6, 7] < [7, 5, 6]: True
(4, 3, 2) < (4, 3, 2): False
------------------------------
[2, 3] == (2, 3): False
[2, 3] != (2, 3): True
------------------------------
[5, 6, 7] < [8]: True
(5, 6, 7) < (8,): True


** Process exited - Return Code: 0 **
```

For a comparison operator to compare two lists or two tuples, Python runs an item-by-item comparison.

https://realpython.com/python-operators-expressions/

Logical Operators



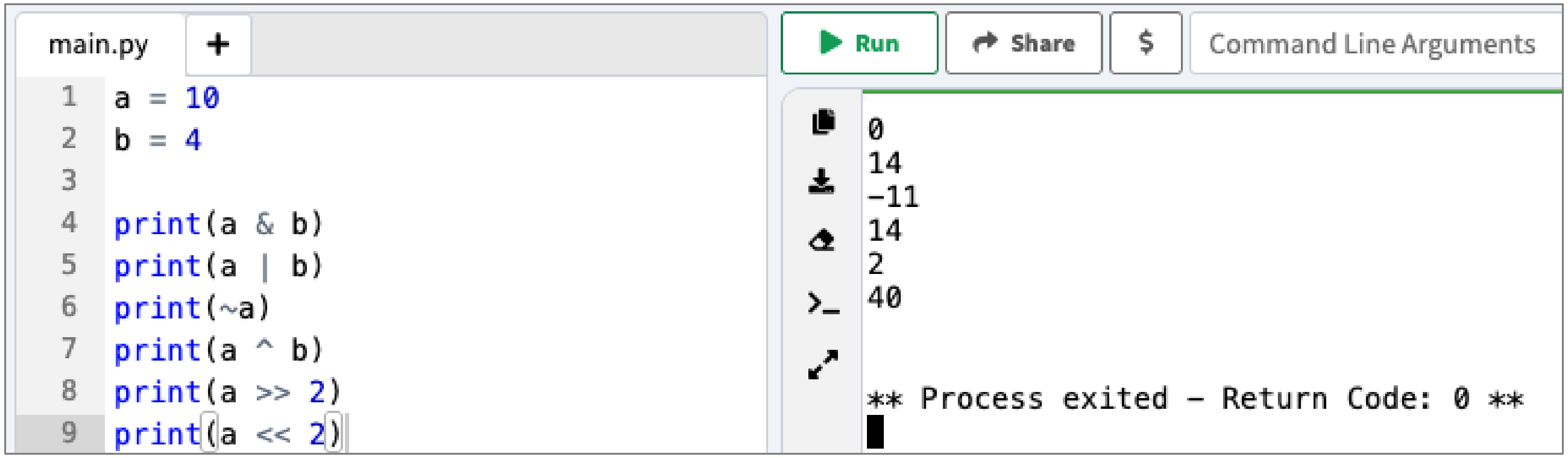
```python
a = True
b = False
print(a and b)
print(a or b)
print(not a)
```

```
False
True
False


** Process exited - Return Code: 0 **
```

Logical operators perform Logical **AND**, Logical **OR** and Logical **NOT** operations. It is used to combine conditional statements.

https://www.geeksforgeeks.org/python/python-operators/

Bitwise Operators



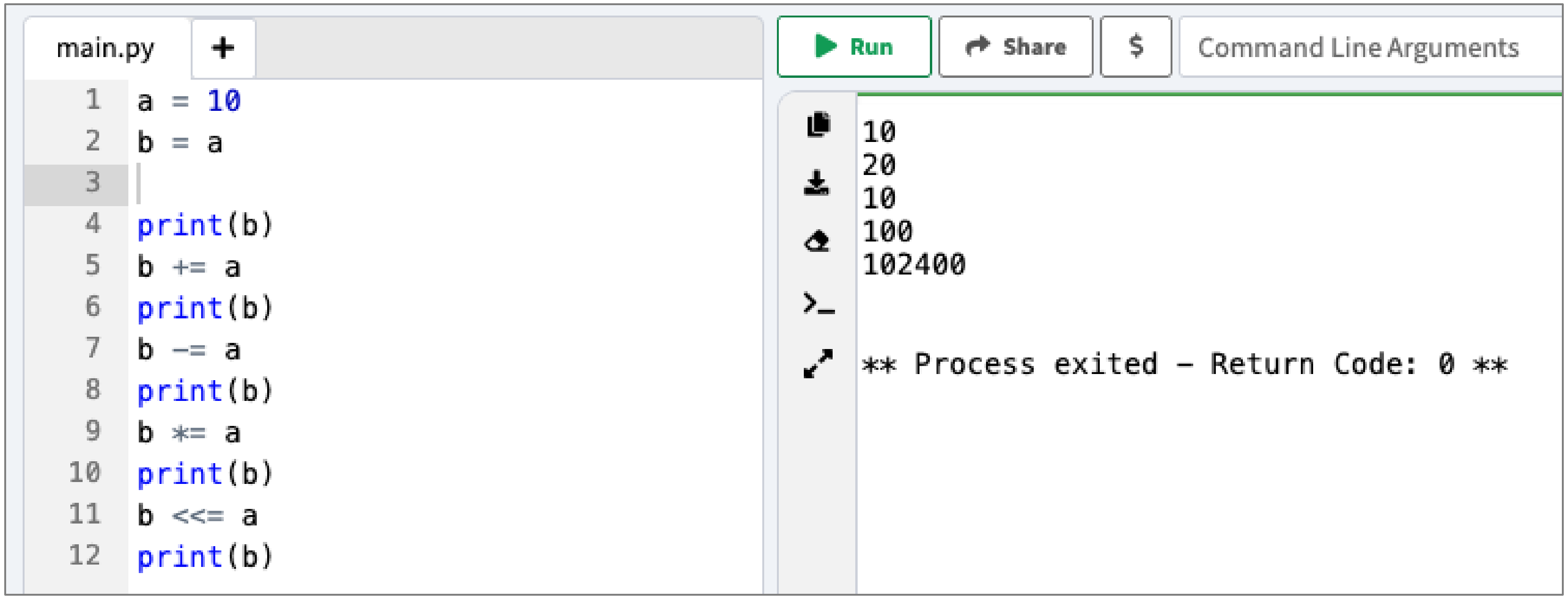
```python
a = 10
b = 4

print(a & b)
print(a | b)
print(~a)
print(a ^ b)
print(a >> 2)
print(a << 2)
```

```
0
14
-11
14
2
40


** Process exited - Return Code: 0 **
```

Bitwise operators act on bits and perform bit-by-bit operations. These are used to operate on binary numbers.

https://www.geeksforgeeks.org/python/python-operators/

Assignment Operators



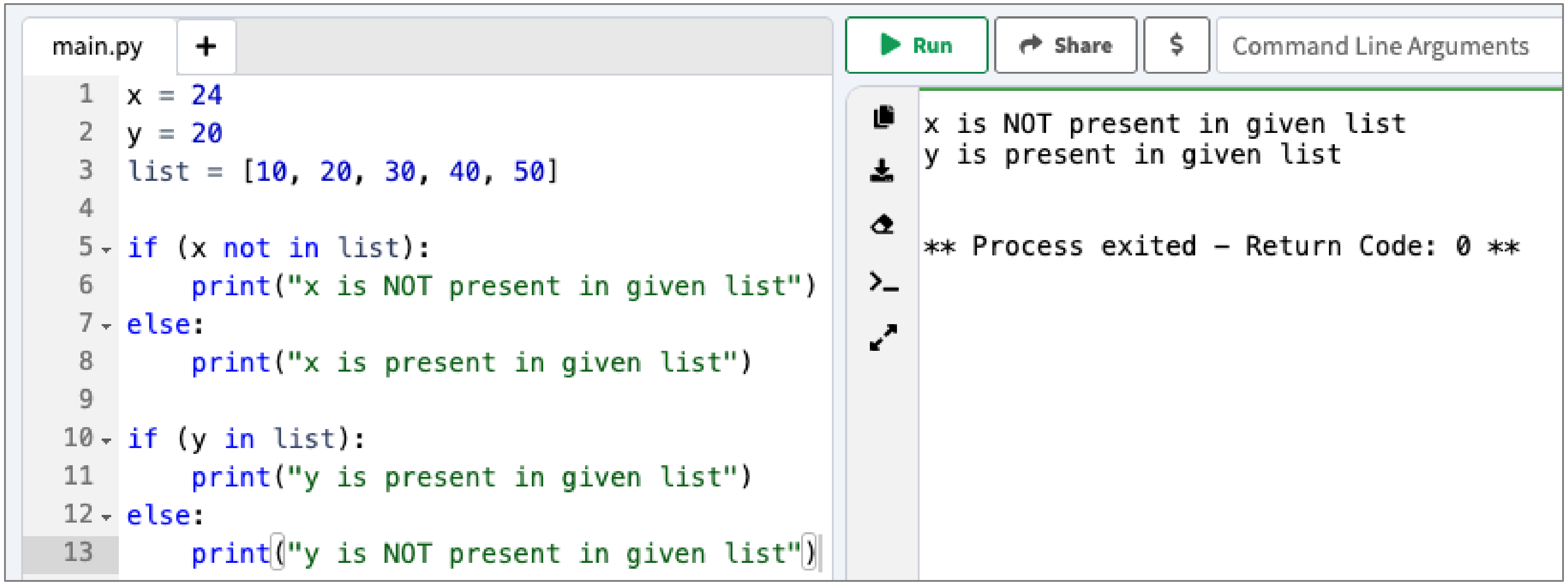
```python
a = 10
b = a

print(b)
b += a
print(b)
b -= a
print(b)
b *= a
print(b)
b <<= a
print(b)
```

```
10
20
10
100
102400


** Process exited - Return Code: 0 **
```

Assignment operators are used to assign values to the variables. This operator is used to assign the value of the right side of the expression to the left side operand.

https://www.geeksforgeeks.org/python/python-operators/

Membership Operators



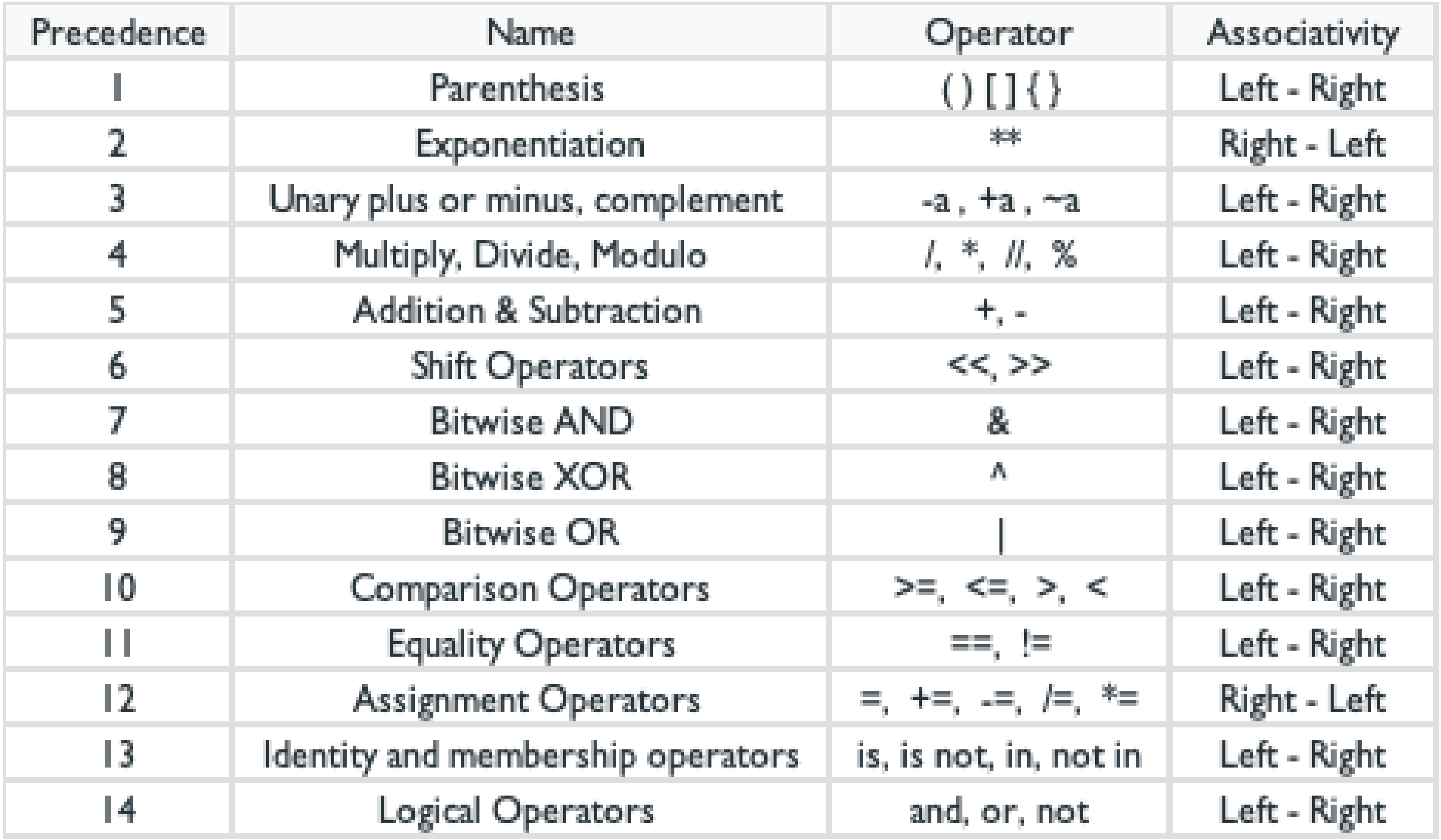
```python
x = 24
y = 20
list = [10, 20, 30, 40, 50]

if (x not in list):
    print("x is NOT present in given list")
else:
    print("x is present in given list")

if (y in list):
    print("y is present in given list")
else:
    print("y is NOT present in given list")
```

```
x is NOT present in given list
y is present in given list


** Process exited - Return Code: 0 **
```

Membership operators: **in** and **not in** are used to test whether a value or variable is in a sequence.

https://www.geeksforgeeks.org/python/python-operators/

Operator Precedence & Associativity



Precedence	Name	Operator	Associativity
1	Parenthesis	( ) [ ] { }	Left - Right
2	Exponentiation	**	Right - Left
3	Unary plus or minus, complement	-a , +a , ~a	Left - Right
4	Multiply, Divide, Modulo	/, *, //, %	Left - Right
5	Addition & Subtraction	+, -	Left - Right
6	Shift Operators	<<, >>	Left - Right
7	Bitwise AND	&	Left - Right
8	Bitwise XOR	^	Left - Right
9	Bitwise OR	|	Left - Right
10	Comparison Operators	>=, <=, >, <	Left - Right
11	Equality Operators	==, !=	Left - Right
12	Assignment Operators	=, +=, -=, /=, *=	Right - Left
13	Identity and membership operators	is, is not, in, not in	Left - Right
14	Logical Operators	and, or, not	Left - Right

Expressions in Practice

- Logical Expressions in Conditions:

```
if (score >= 5) and (absence < 3)
```

- Expressions in Loops:

```
while x < 10:
```

- Expressions in List Comprehension

```
[x**2 for x in range(10) if x % 2 == 0]
```

- Expressions in Functions

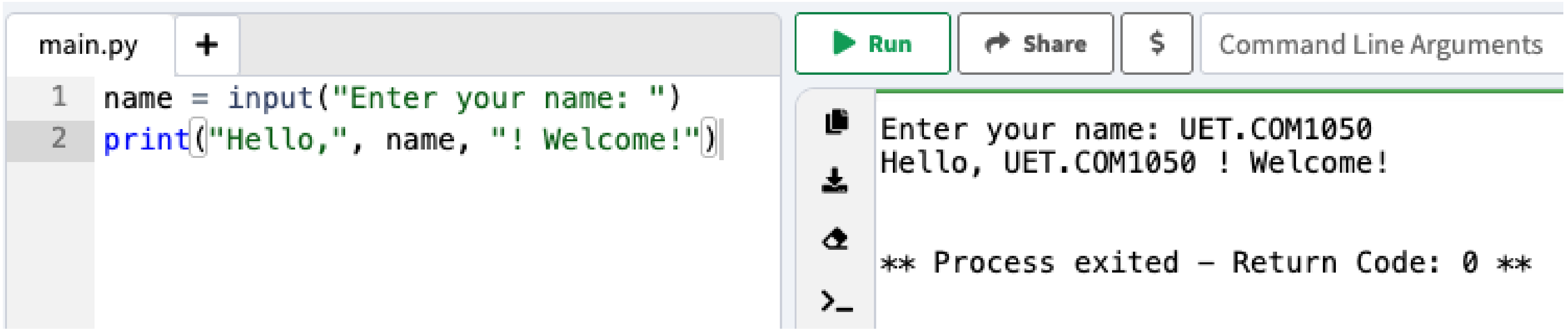
```
def square(x):
    return x**2
```

- Nested Expressions:

```
max(a, b, c) > 10 and (a + b) % 2 == 0
```

Input & Output in Python

Taking Input in Python



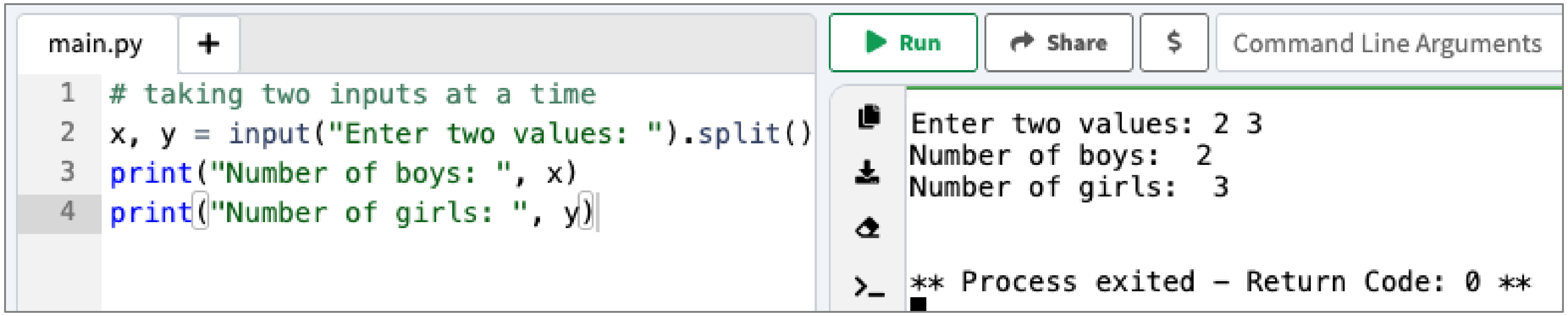
```python
name = input("Enter your name: ")
print("Hello,", name, "! Welcome!")
```

```
Enter your name: UET.COM1050
Hello, UET.COM1050 ! Welcome!


** Process exited - Return Code: 0 **
```

Python's input() function is used to take user input. By default, it returns the user input in form of a string.

https://www.geeksforgeeks.org/python/input-and-output-in-python/

Taking Multiple Input in Python



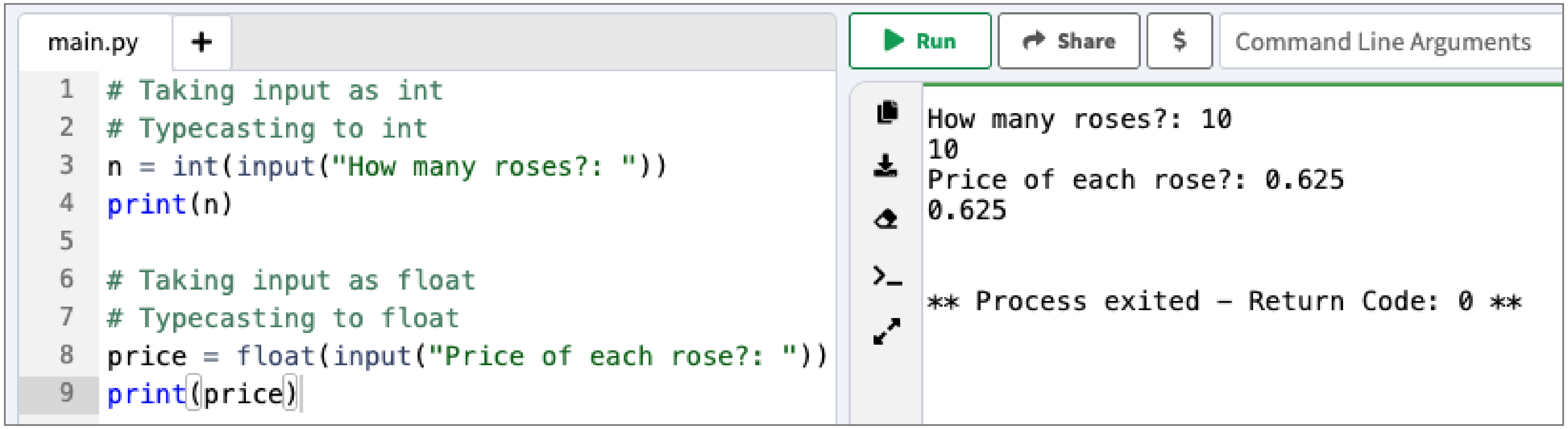
```python
# taking two inputs at a time
x, y = input("Enter two values: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)
```

```
Enter two values: 2 3
Number of boys:  2
Number of girls:  3


** Process exited - Return Code: 0 **
```

Multiple input is split into separate variables for each value using the split() method

https://www.geeksforgeeks.org/python/input-and-output-in-python/

Input Type Casting



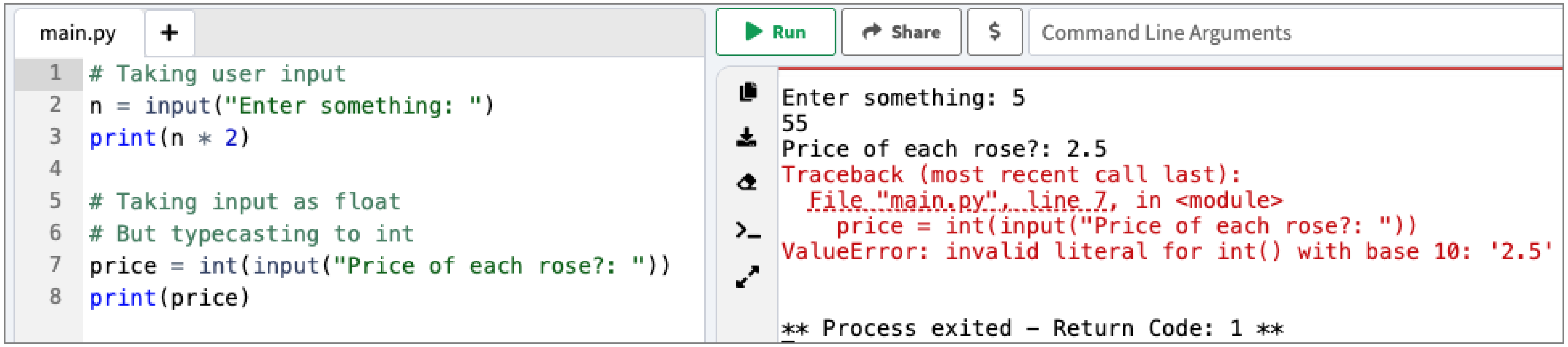
```python
# Taking input as int
# Typecasting to int
n = int(input("How many roses?: "))
print(n)

# Taking input as float
# Typecasting to float
price = float(input("Price of each rose?: "))
print(price)
```

```
How many roses?: 10
10
Price of each rose?: 0.625
0.625


** Process exited - Return Code: 0 **
```

By default input() function helps in taking user input as string, which might be casted to int/float

https://www.geeksforgeeks.org/python/input-and-output-in-python/

Handling User Input: Common Mistakes



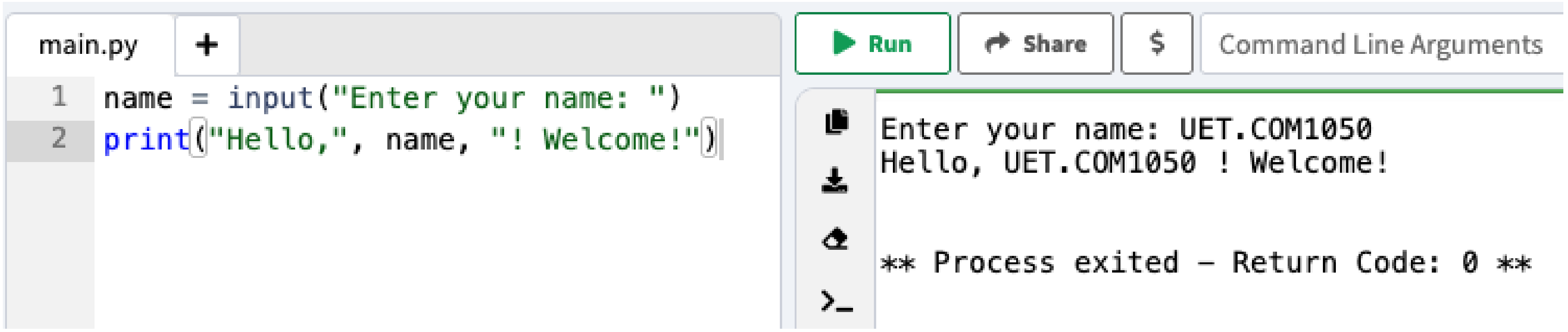
```python
# Taking user input
n = input("Enter something: ")
print(n * 2)

# Taking input as float
# But typecasting to int
price = int(input("Price of each rose?: "))
print(price)
```

```
Enter something: 5
55
Price of each rose?: 2.5
Traceback (most recent call last):
  File "main.py", line 7, in <module>
    price = int(input("Price of each rose?: "))
ValueError: invalid literal for int() with base 10: '2.5'


** Process exited - Return Code: 1 **
```

By default input() function helps in taking user input as string, you forget to use an appropriate typecasting

https://www.geeksforgeeks.org/python/input-and-output-in-python/

Printing Output in Python



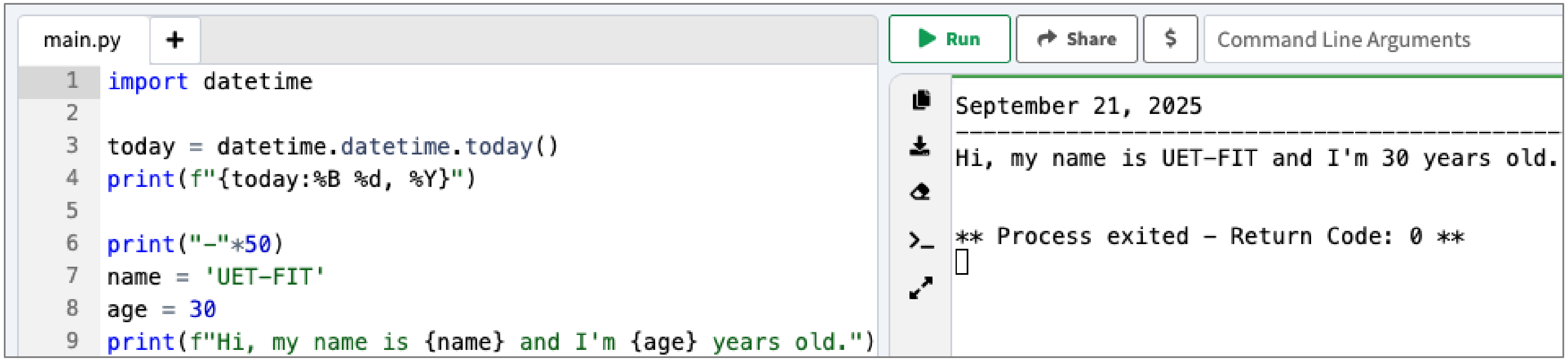
```python
name = input("Enter your name: ")
print("Hello,", name, "! Welcome!")
```

```
Enter your name: UET.COM1050
Hello, UET.COM1050 ! Welcome!


** Process exited - Return Code: 0 **
```

Python use the print() function to display text, variables and expressions on the console.

https://www.geeksforgeeks.org/python/input-and-output-in-python/

Printing Output with f-string in Python



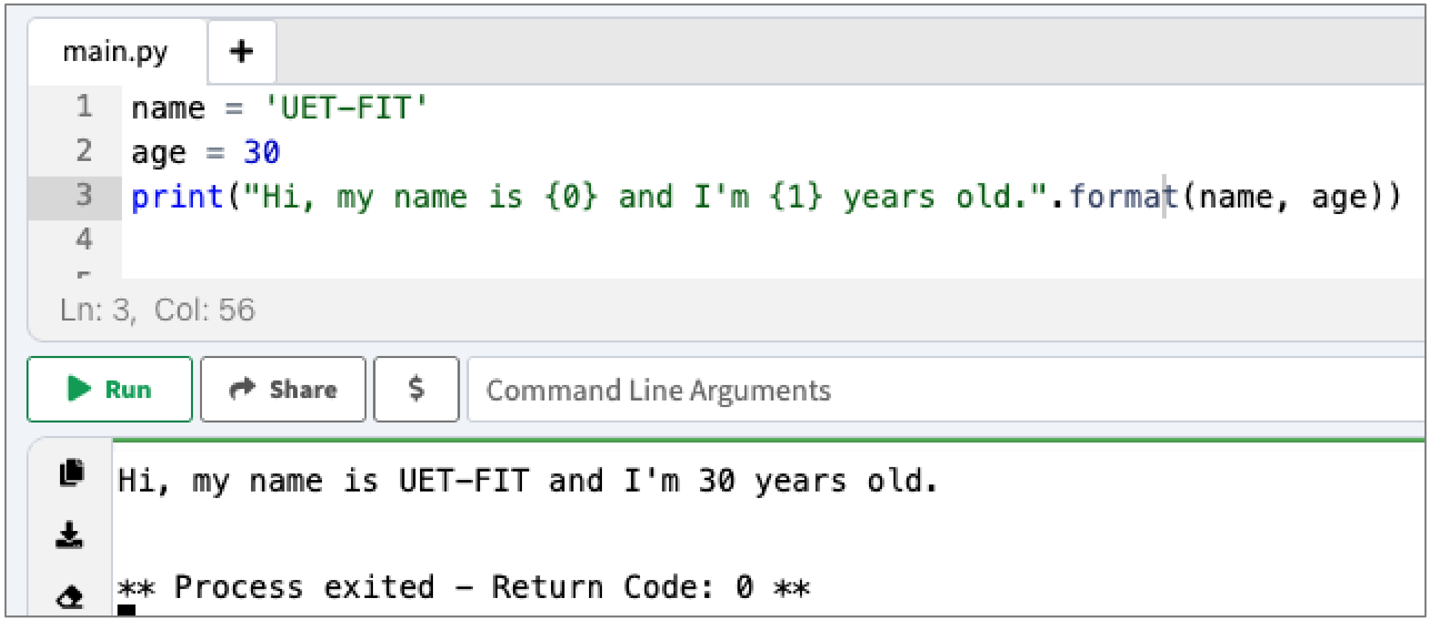
```python
import datetime

today = datetime.datetime.today()
print(f"{today:%B %d, %Y}")

print("-"*50)
name = 'UET-FIT'
age = 30
print(f"Hi, my name is {name} and I'm {age} years old.")
```

```
September 21, 2025
--------------------------------------------------
Hi, my name is UET-FIT and I'm 30 years old.


** Process exited - Return Code: 0 **
```

Python introduces f-strings (formatted string literals) to make string formatting and interpolation easier.

https://www.geeksforgeeks.org/python/formatted-string-literals-f-strings-python/

Printing Output with .format() in Python



```python
name = 'UET-FIT'
age = 30
print("Hi, my name is {0} and I'm {1} years old.".format(name, age))
```

```
Hi, my name is UET-FIT and I'm 30 years old.


** Process exited - Return Code: 0 **
```

Python use .format() method to create formatted strings by embedding variables or values into placeholders within a template string.

https://www.geeksforgeeks.org/python/python-string-format-method/

Summary - Key Takeaways

- **Variables** are **names** used to store data.
- **Data types** define the **kind of value** a variable can hold and **operators can perform** on it.
  - It could be changed during program execution
- **An expression** combines **operators** and **operands** to produce a value.
- **Operators** perform specific **actions** on **variables** and **values**.
- **Input** is taken as a **string**. Python use **print()** to print **output** with configurable format.