PROGRAMMING IN PYTHON
MCA-161A
4 Credits (3-0-2)
MCA 5th Sem (2020-21)

R K Dwivedi
Assistant Professor
Department of ITCA
MMMUT Gorakhpur

Contents

Programming Basics and Decision Making

1. Introduction:

Key features and applications of Python,

Python Editors and Compilers (Interpreters),

Using different offline and online Python IDE,

Interacting with Python programs

2. Data types:

Numeric, Boolean, Strings, Lists, Tuples, Sets, Dictionary

3. Variables:

Declaration and initialization

4. Simple Statements:

Taking inputs from user, Displaying outputs

5. Other concepts:

Operators, Expressions, Indentation, Comments, Casting

6. Conditional statements:

if…elif…else

1. Introduction

1. Introduction

Key features

- It is **most widely used, high-level, general-purpose, multi-purpose and interpreted** programming language.
- There are two major Python versions: **Python 2 and Python 3 (latest version is 3.7.1, we can simply call it as Python 3).**
- It was created by **Guido Van Rossum**, and released in **1991**. **Python 3.0 was released in 2008.**
- **It is developed under an OSI-approved open source license, making it freely usable and distributable, even for commercial use.**
- Python works on different **platforms (Windows, Mac, Linux, Raspberry Pi, etc)**. **It is portable across operating systems.**
- Python has a **simple syntax** similar to the English language.
- Python runs on an **interpreter** system, meaning that code can be **executed as soon as it is written**.
- Python can be treated in a **procedural** way, an **object-orientated** way or a **functional way**.
- Its design philosophy **emphasizes code readability**.
- **I**ts syntax allows programmers to express concepts in **fewer lines of code**.
- The best way we learn anything is by **practice** and **exercise** questions.
- Python uses **new lines** to complete a command **instead of semicolons or parentheses**.
- Python uses **indentation** through **whitespace** instead of **curly-brackets**, to define the scope of loops, functions and classes.
- Python can connect to **database** systems. It can also **read and modify files**.
- **It is very well suited for beginners and also for experienced programmers with other programming languages like C++ and Java.**
- **The biggest strength of Python is huge collection of standard libraries that can be required in many applications (NumPy for numerical calculations, Pandas for data analytics etc).**
- you can download it for free from the following website: **https://www.python.org/**

1. Introduction

Applications of Python

- Python can be used on a server to create **web applications**.
- **It can be used to create GUI based desktop applications(Games, Scientific and Business Applications).**
- **It is also used to create test frameworks and multimedia applications.**
- **It is used to develop operating systems and programming language.**
- It can be used to handle **image processing, text processing and natural language processing.**
- It can be used to create programs for **machine learning, deep learning, data science, big data** and **data analytics applications**.
- It can also perform **complex mathematics along with all cutting edge technology in software industry**.

Organizations and tech-giant companies using Python :

1) Google(Components of Google spider and Search Engine)
2) Yahoo(Maps)
3) YouTube
4) Mozilla
5) Dropbox
6) Microsoft
7) Cisco
8) Spotify
9) Quora
10) Instagram
11)Amazon
12)Facebook
13)Uber etc.

1. Introduction

Python Editors and Interpreters

- **Python is interpreted language i.e it's not compiled and the interpreter will check the code line by line.**
- Code can be written in a **text editor** with **.py** extension and then it can be put into the **python interpreter** for execution.

1. Introduction

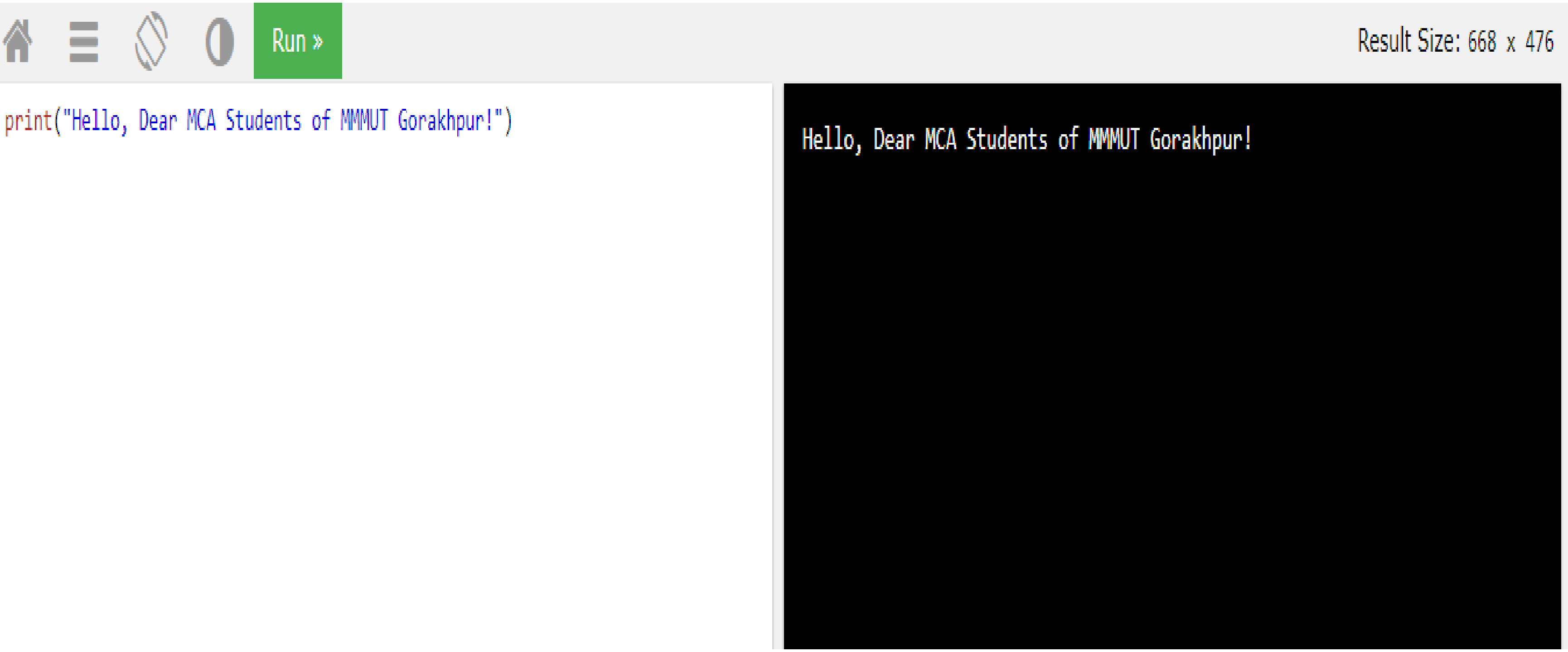
Using different offline and online Python IDE

Offline IDE: Thonny, Pycharm, Spyder, IDLE, Netbeans or Eclipse **PyDev etc.**

Online IDE:
https://www.w3schools.com/python/
https://www.w3resource.com/python/python-tutorial.php
https://www.geeksforgeeks.org/python-programming-language/
https://www.tutorialspoint.com/execute_python_online.php

1. Introduction

Interacting with Python programs



```
print("Hello, Dear MCA Students of MMMUT Gorakhpur!")
```

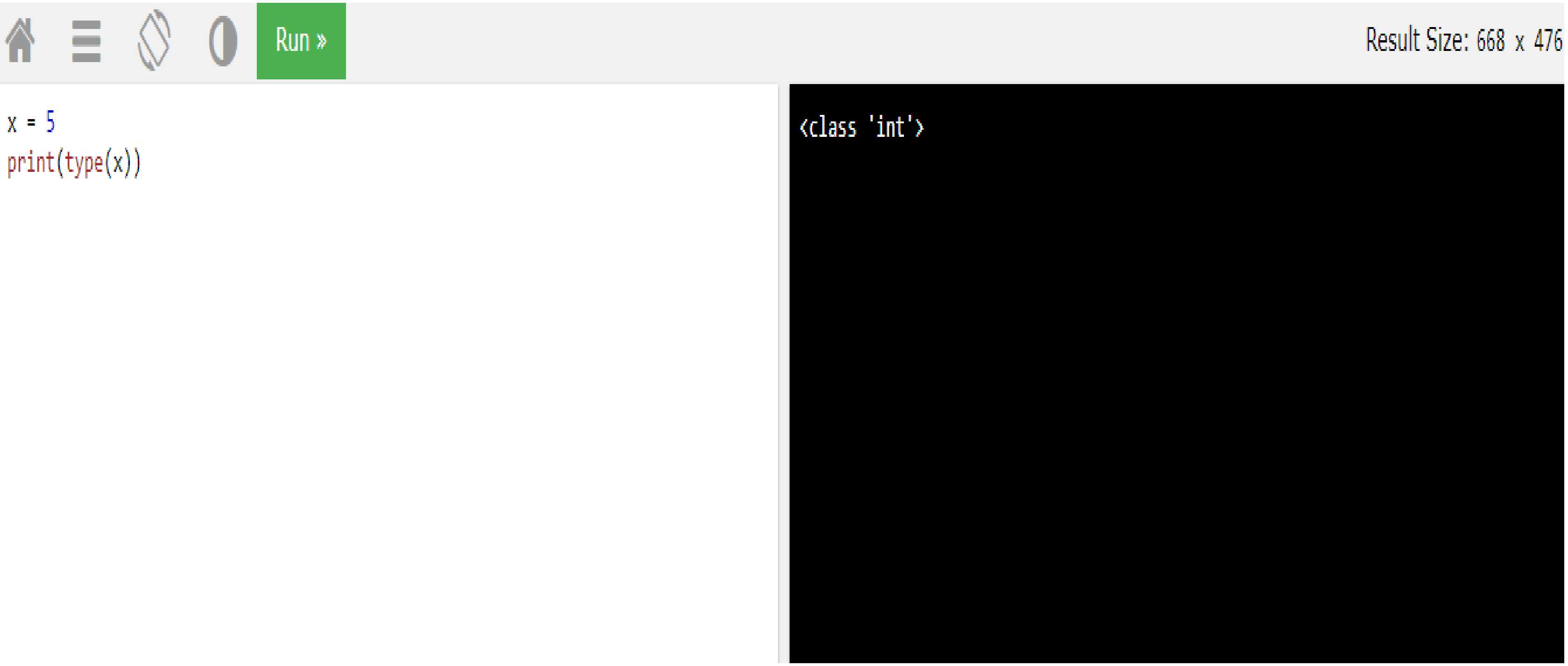
```
Hello, Dear MCA Students of MMMUT Gorakhpur!
```

2. Datatypes

2. Data types

Python is **dynamically typed language**(No need to mention data type based on value assigned, it takes data type)

You can get the data type of any object by using the type( ) function

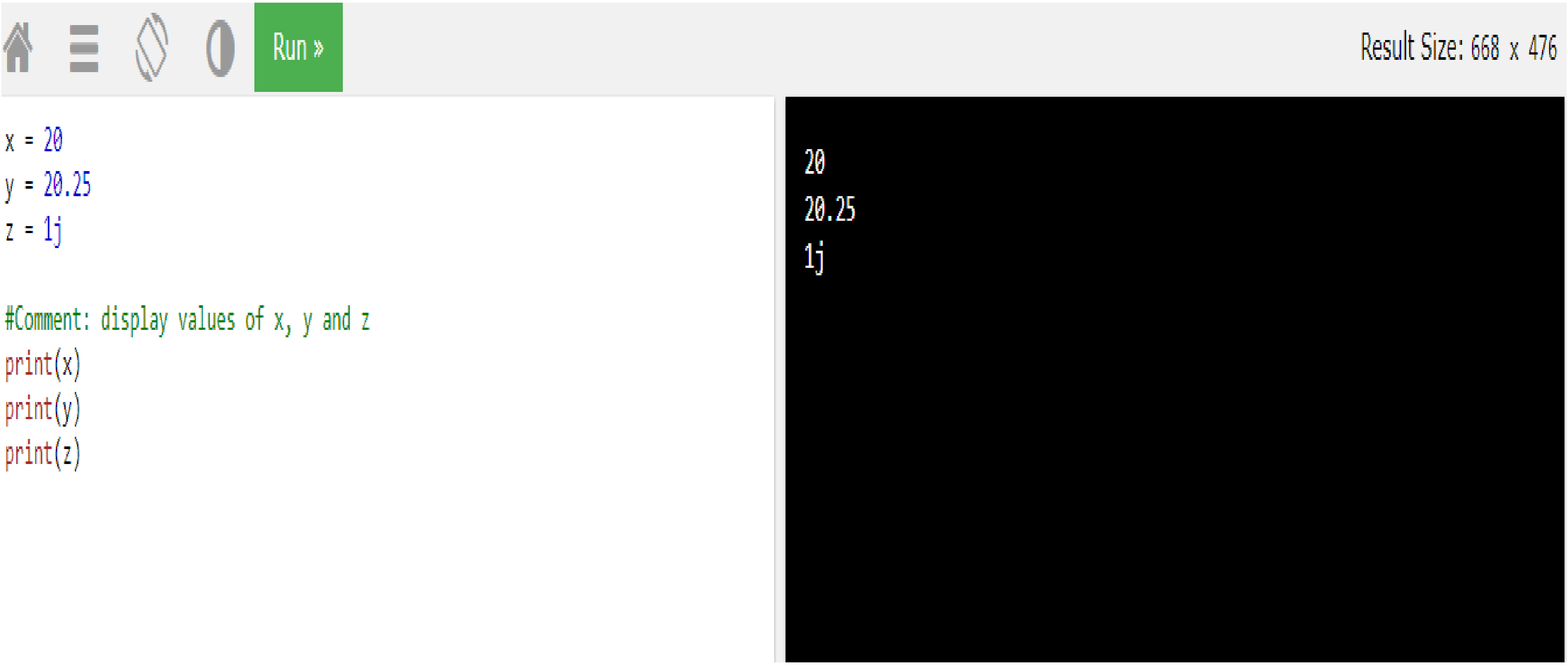


```
x = 5
print(type(x))
```

```
<class 'int'>
```

2. Data types

Numeric Type: int, float, complex

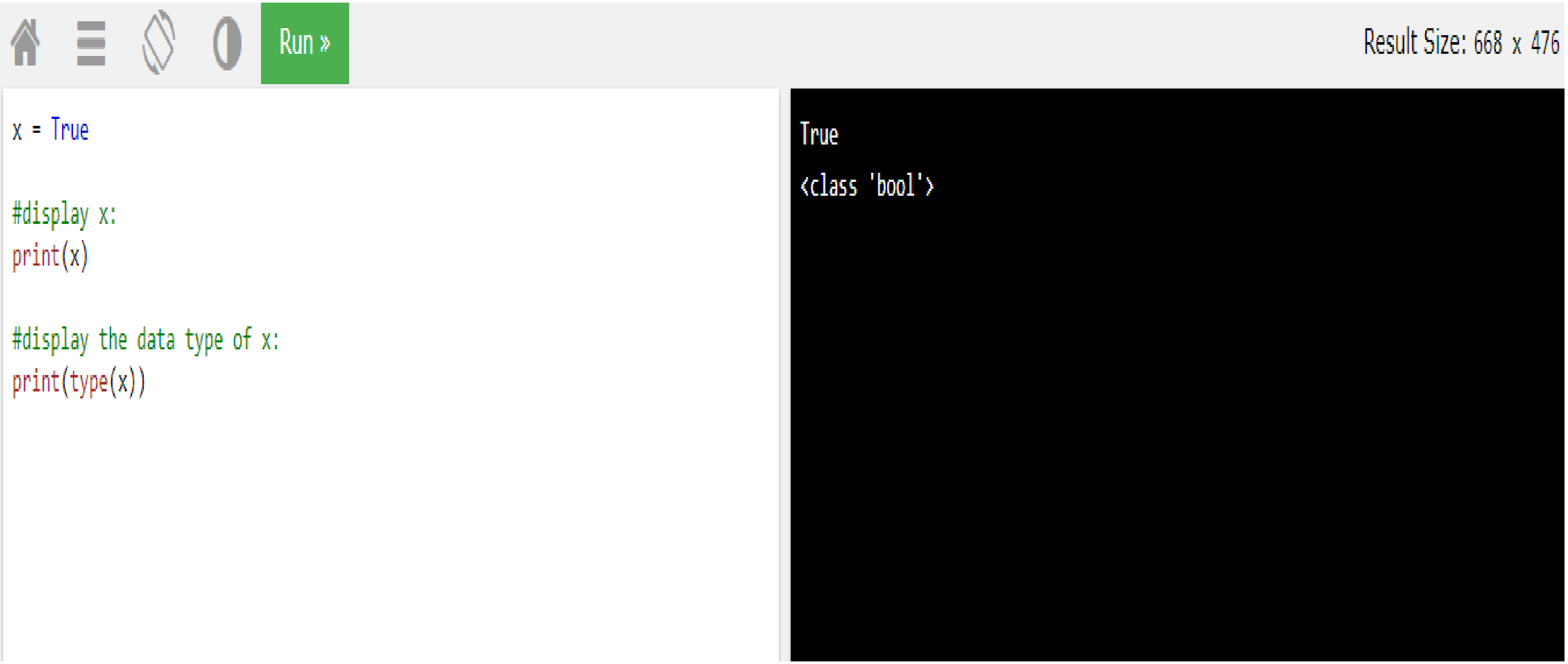


```
x = 20
y = 20.25
z = 1j

#Comment: display values of x, y and z
print(x)
print(y)
print(z)
```

```
20
20.25
1j
```

2. Data types

Boolean Type: bool



```
x = True

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
True
<class 'bool'>
```

2. Data types

Text Type (Strings): str

We can assign a string to a variable as below:

```
a = "Hello“
```

We can assign a **multiline string** to a variable by using **three single or three double quotes** as below:
(**Note:** in the result, the line breaks are inserted at the same position as in the code.):

```
a = ’‘’Hi everyone,
    This is Python Class.’‘’
a = ”””Hi everyone,
    This is Python Class.”””
```

To **concatenate**, or combine, two strings we can use the **+ operator as below**:

```
a = "Hello"
b = “Class"
c = a + " " + b
```

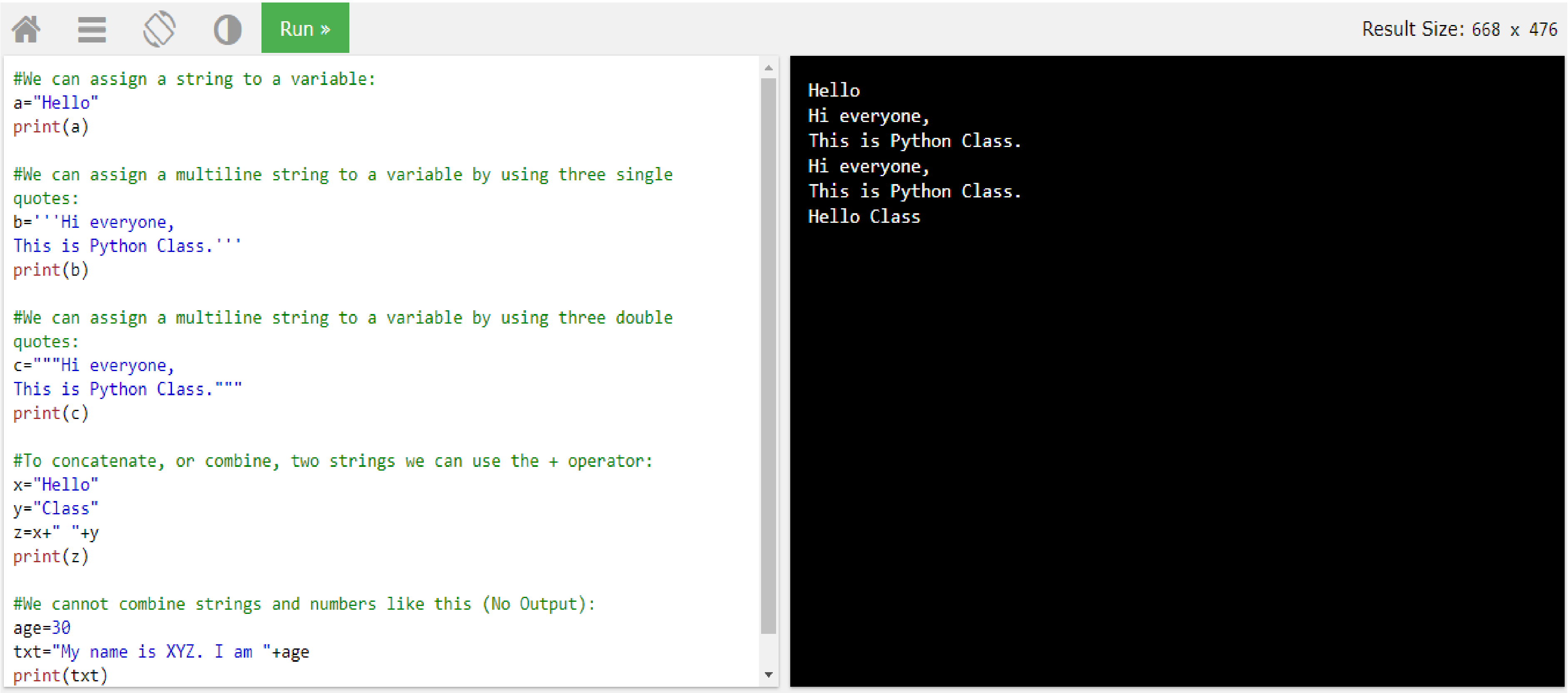
We cannot combine **strings** and **numbers** like this:

```
age = 30
txt = "My name is XYZ, I am " + age
```

2. Data types

Text Type (Strings): str

…continued



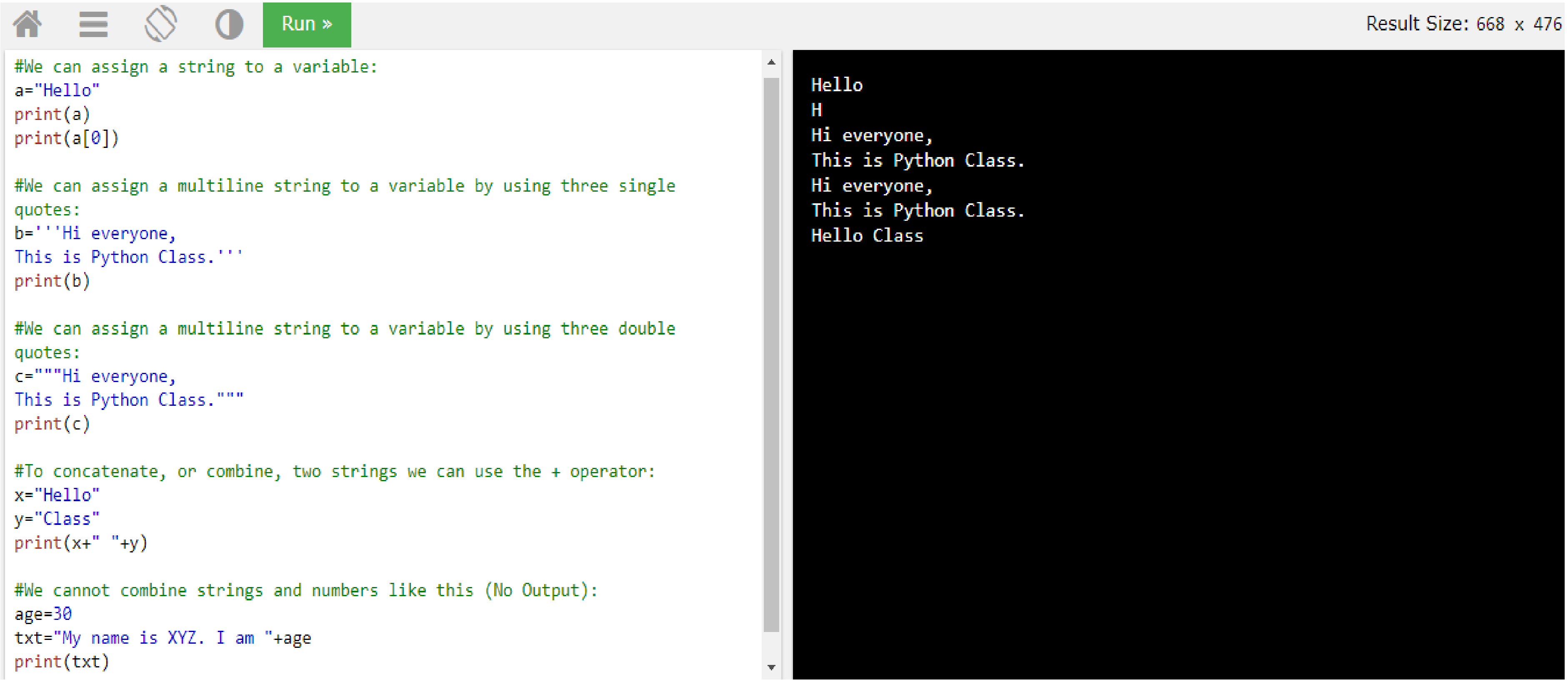
```
#We can assign a string to a variable:
a="Hello"
print(a)

#We can assign a multiline string to a variable by using three single
quotes:
b='''Hi everyone,
This is Python Class.'''
print(b)

#We can assign a multiline string to a variable by using three double
quotes:
c="""Hi everyone,
This is Python Class."""
print(c)

#To concatenate, or combine, two strings we can use the + operator:
x="Hello"
y="Class"
z=x+" "+y
print(z)

#We cannot combine strings and numbers like this (No Output):
age=30
txt="My name is XYZ. I am "+age
print(txt)
```

```
Hello
Hi everyone,
This is Python Class.
Hi everyone,
This is Python Class.
Hello Class
```

2. Data types

Text Type (Strings): str

…continued



```
#We can assign a string to a variable:
a="Hello"
print(a)
print(a[0])

#We can assign a multiline string to a variable by using three single
quotes:
b='''Hi everyone,
This is Python Class.'''
print(b)

#We can assign a multiline string to a variable by using three double
quotes:
c="""Hi everyone,
This is Python Class."""
print(c)

#To concatenate, or combine, two strings we can use the + operator:
x="Hello"
y="Class"
print(x+" "+y)

#We cannot combine strings and numbers like this (No Output):
age=30
txt="My name is XYZ. I am "+age
print(txt)
```

```
Hello
H
Hi everyone,
This is Python Class.
Hi everyone,
This is Python Class.
Hello Class
```

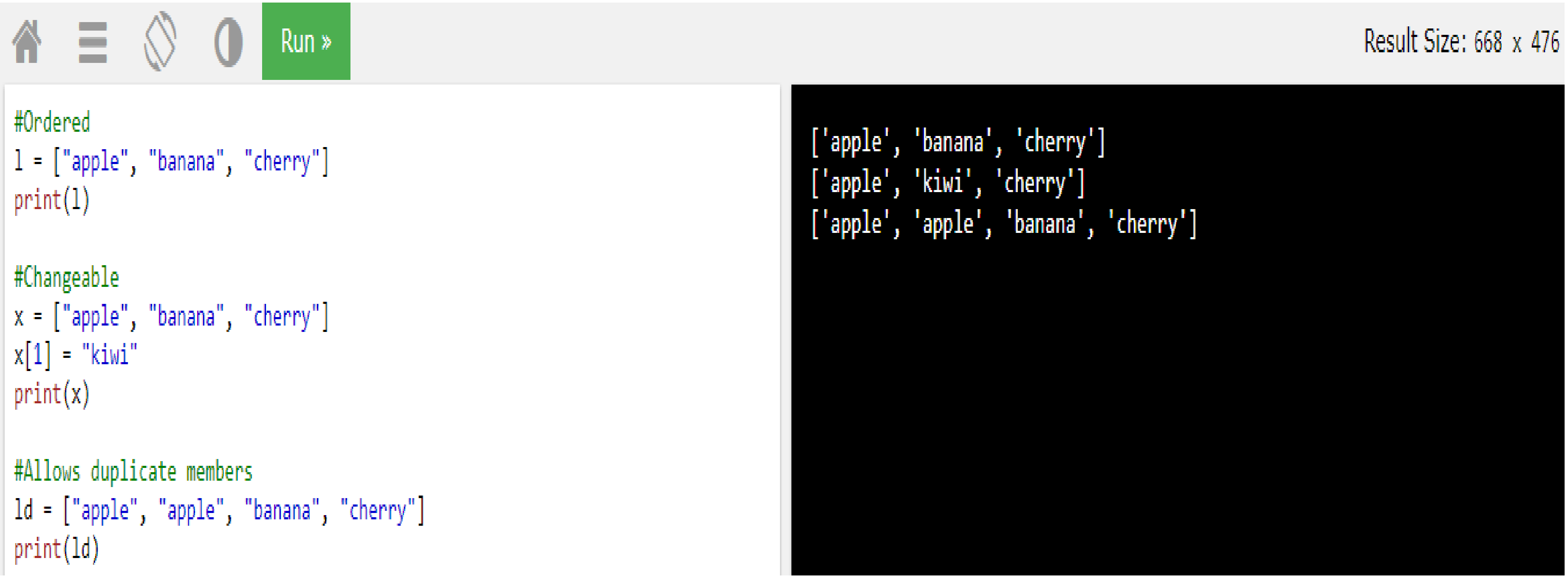
2. Data types

Sequence Type: list

List is a collection which is **ordered, indexed** and **changeable**. It **allows duplicate** members.

Lists are written with square brackets [ ].

We can access list items by referring to the index number, inside square brackets.



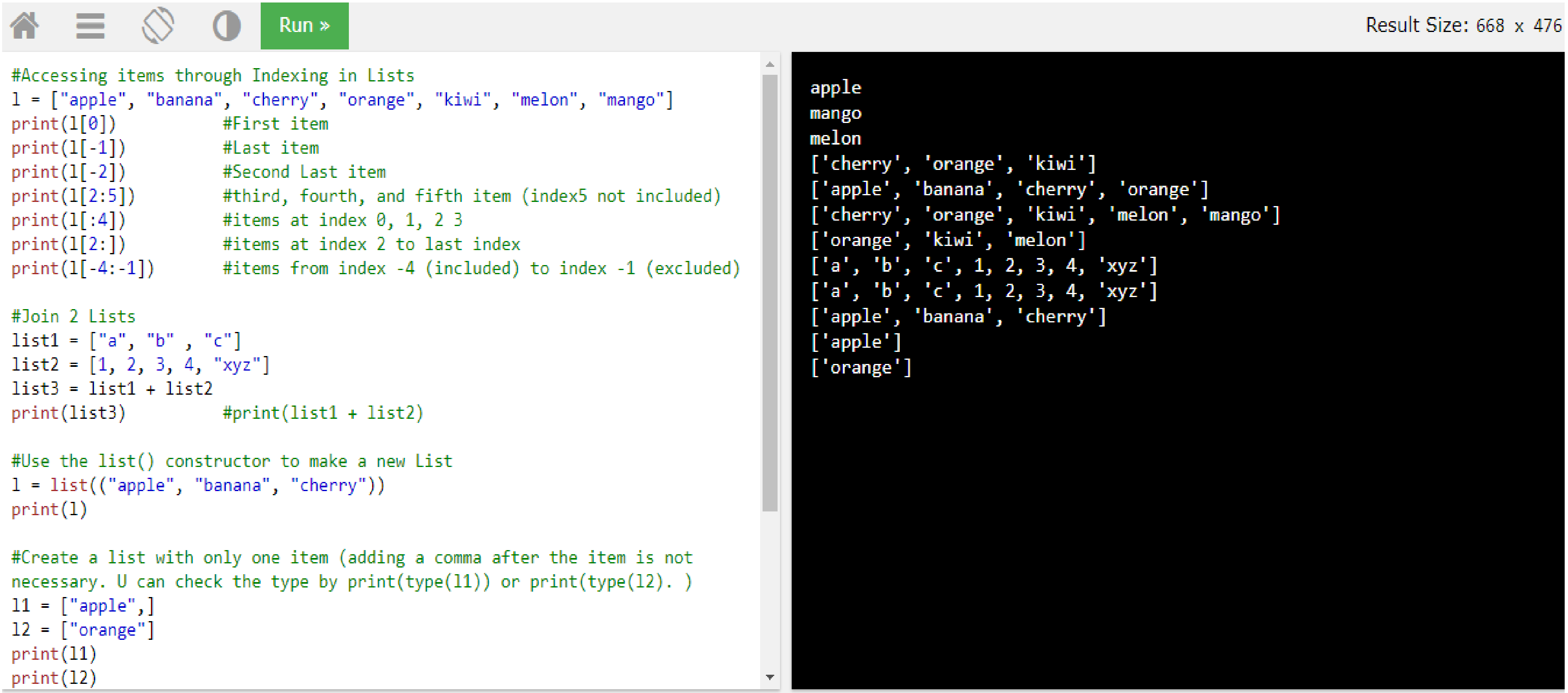
```
#Ordered
l = ["apple", "banana", "cherry"]
print(l)

#Changeable
x = ["apple", "banana", "cherry"]
x[1] = "kiwi"
print(x)

#Allows duplicate members
ld = ["apple", "apple", "banana", "cherry"]
print(ld)
```

```
['apple', 'banana', 'cherry']
['apple', 'kiwi', 'cherry']
['apple', 'apple', 'banana', 'cherry']
```

2. Data types

Sequence Type: list

…continued



```python
#Accessing items through Indexing in Lists
l = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(l[0])           #First item
print(l[-1])          #Last item
print(l[-2])          #Second Last item
print(l[2:5])         #third, fourth, and fifth item (index5 not included)
print(l[:4])          #items at index 0, 1, 2 3
print(l[2:])          #items at index 2 to last index
print(l[-4:-1])       #items from index -4 (included) to index -1 (excluded)

#Join 2 Lists
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3, 4, "xyz"]
list3 = list1 + list2
print(list3)          #print(list1 + list2)

#Use the list() constructor to make a new List
l = list(("apple", "banana", "cherry"))
print(l)

#Create a list with only one item (adding a comma after the item is not
necessary. U can check the type by print(type(l1)) or print(type(l2). )
l1 = ["apple",]
l2 = ["orange"]
print(l1)
print(l2)
```

```
apple
mango
melon
['cherry', 'orange', 'kiwi']
['apple', 'banana', 'cherry', 'orange']
['cherry', 'orange', 'kiwi', 'melon', 'mango']
['orange', 'kiwi', 'melon']
['a', 'b', 'c', 1, 2, 3, 4, 'xyz']
['a', 'b', 'c', 1, 2, 3, 4, 'xyz']
['apple', 'banana', 'cherry']
['apple']
['orange']
```

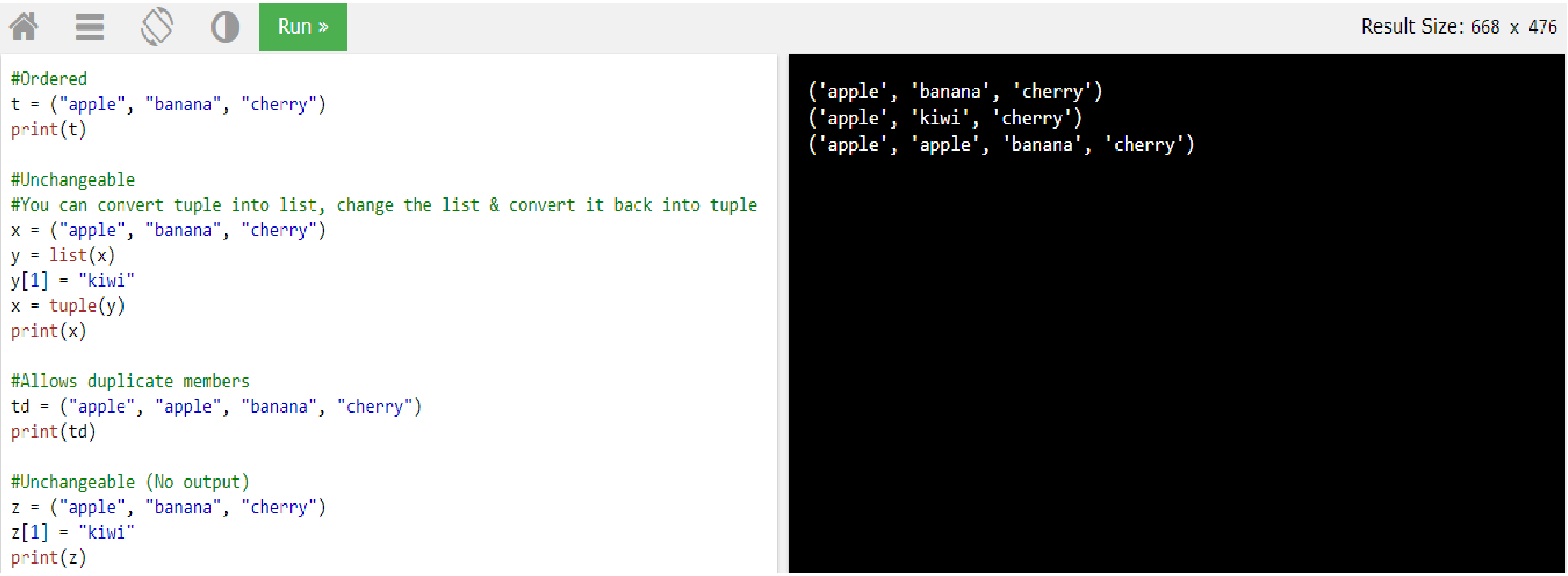
2. Data types

Sequence Type: tuple

Tuple is a collection which is ordered, indexed and unchangeable. It allows duplicate members.

Tuples are written with **round brackets ( )**.

We can access tuple items by referring to the index number, inside square brackets.



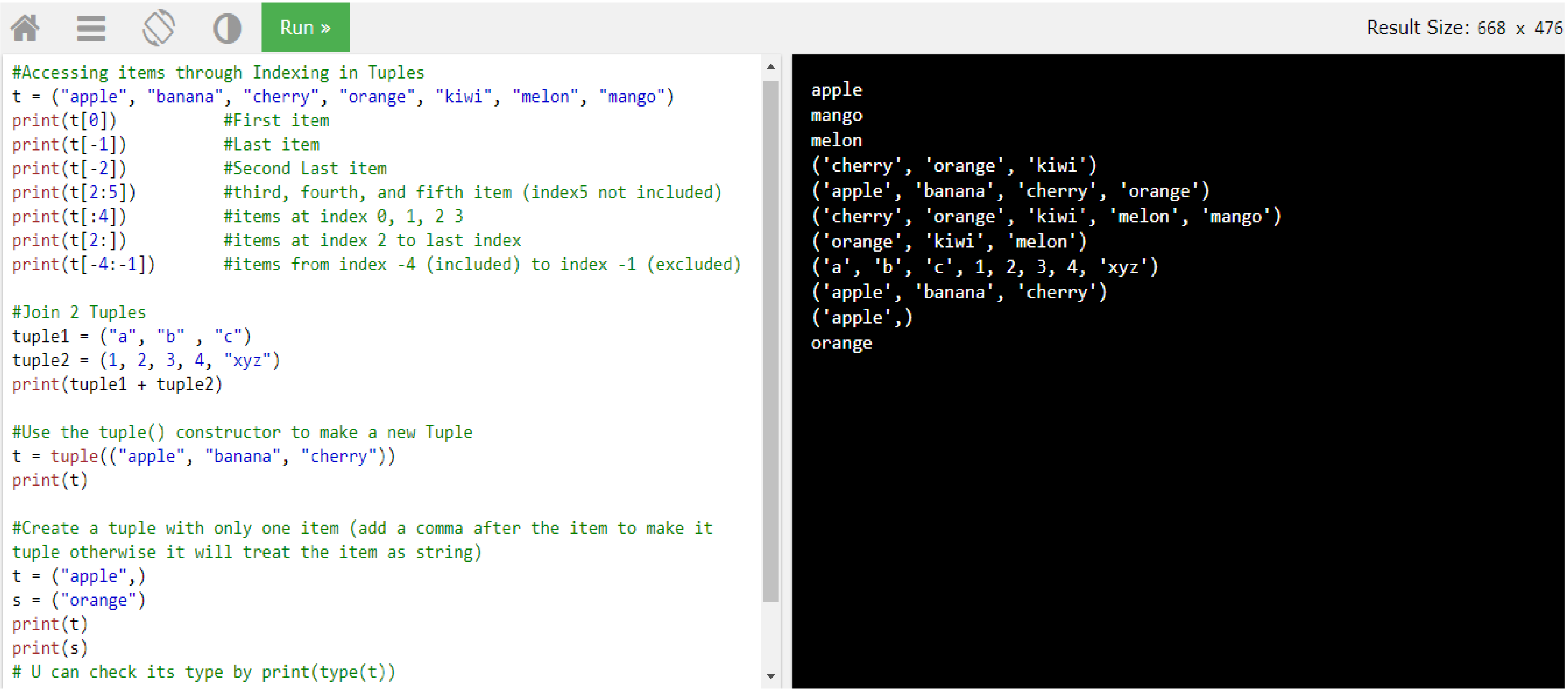
```
#Ordered
t = ("apple", "banana", "cherry")
print(t)

#Unchangeable
#You can convert tuple into list, change the list & convert it back into tuple
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)

#Allows duplicate members
td = ("apple", "apple", "banana", "cherry")
print(td)

#Unchangeable (No output)
z = ("apple", "banana", "cherry")
z[1] = "kiwi"
print(z)
```

```
('apple', 'banana', 'cherry')
('apple', 'kiwi', 'cherry')
('apple', 'apple', 'banana', 'cherry')
```

2. Data types

Sequence Type: tuple

…continued



```
#Accessing items through Indexing in Tuples
t = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(t[0])          #First item
print(t[-1])         #Last item
print(t[-2])         #Second Last item
print(t[2:5])        #third, fourth, and fifth item (index5 not included)
print(t[:4])         #items at index 0, 1, 2 3
print(t[2:])         #items at index 2 to last index
print(t[-4:-1])      #items from index -4 (included) to index -1 (excluded)

#Join 2 Tuples
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3, 4, "xyz")
print(tuple1 + tuple2)

#Use the tuple() constructor to make a new Tuple
t = tuple(("apple", "banana", "cherry"))
print(t)

#Create a tuple with only one item (add a comma after the item to make it
tuple otherwise it will treat the item as string)
t = ("apple",)
s = ("orange")
print(t)
print(s)
# U can check its type by print(type(t))
```

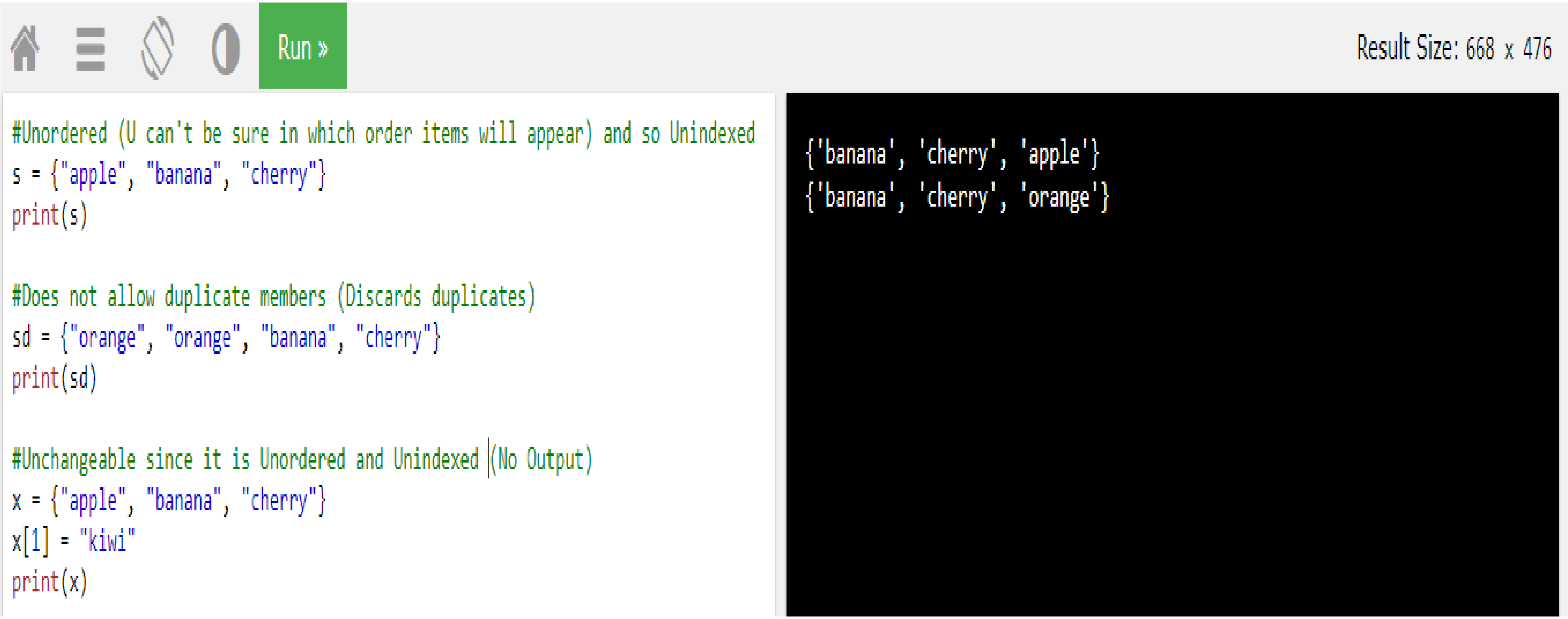
```
apple
mango
melon
('cherry', 'orange', 'kiwi')
('apple', 'banana', 'cherry', 'orange')
('cherry', 'orange', 'kiwi', 'melon', 'mango')
('orange', 'kiwi', 'melon')
('a', 'b', 'c', 1, 2, 3, 4, 'xyz')
('apple', 'banana', 'cherry')
('apple',)
orange
```

2. Data types

Set Type: set

Set is a collection which is unordered, unindexed and unchangeable. It does not allow duplicate members.

Sets are written with **curly brackets { }**.



```
#Unordered (U can't be sure in which order items will appear) and so Unindexed
s = {"apple", "banana", "cherry"}
print(s)

#Does not allow duplicate members (Discards duplicates)
sd = {"orange", "orange", "banana", "cherry"}
print(sd)

#Unchangeable since it is Unordered and Unindexed (No Output)
x = {"apple", "banana", "cherry"}
x[1] = "kiwi"
print(x)
```

```
{'banana', 'cherry', 'apple'}
{'banana', 'cherry', 'orange'}
```

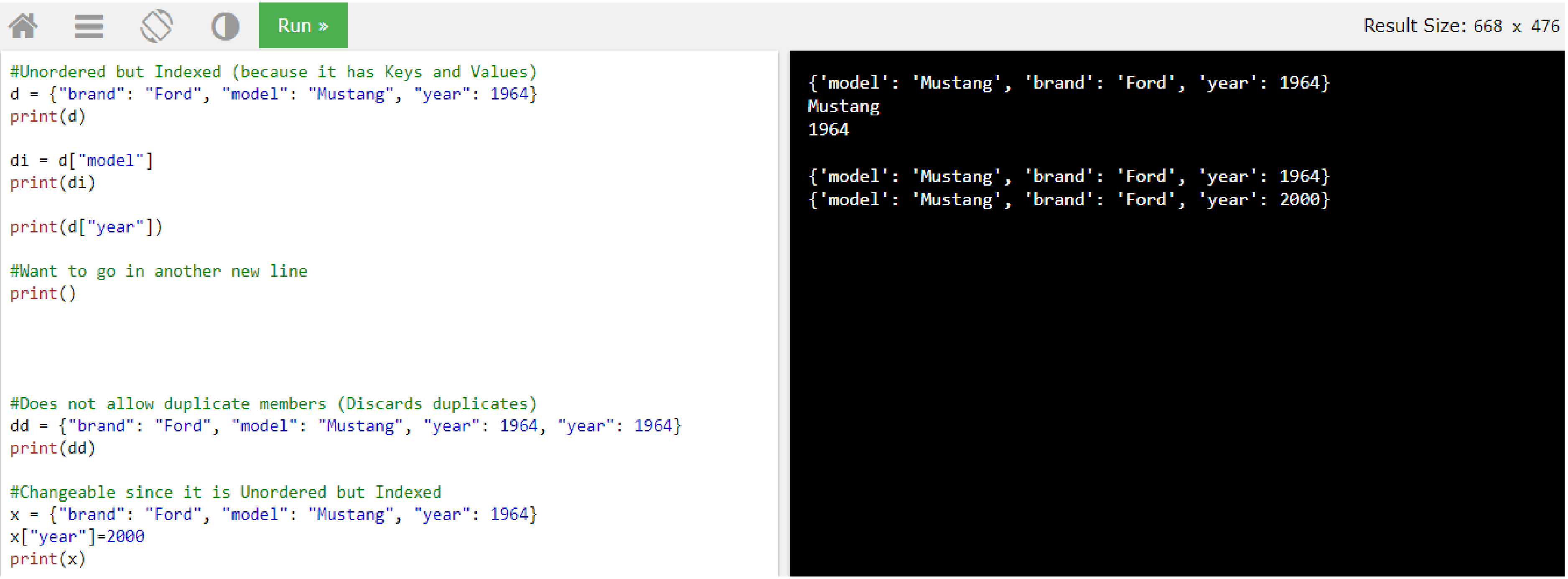
2. Data types

Mapping Type (Dictionary): dict

Dictionary is a collection which is unordered, indexed and changeable. It does not allow duplicate members.

Dictionaries are written with curly brackets, and they have keys and values.

We can access the items of a dictionary by referring to its **key name**, inside **square brackets.**



```
#Unordered but Indexed (because it has Keys and Values)
d = {"brand": "Ford", "model": "Mustang", "year": 1964}
print(d)

di = d["model"]
print(di)

print(d["year"])

#Want to go in another new line
print()




#Does not allow duplicate members (Discards duplicates)
dd = {"brand": "Ford", "model": "Mustang", "year": 1964, "year": 1964}
print(dd)

#Changeable since it is Unordered but Indexed
x = {"brand": "Ford", "model": "Mustang", "year": 1964}
x["year"]=2000
print(x)
```

```
{'model': 'Mustang', 'brand': 'Ford', 'year': 1964}
Mustang
1964

{'model': 'Mustang', 'brand': 'Ford', 'year': 1964}
{'model': 'Mustang', 'brand': 'Ford', 'year': 2000}
```

3. Variables

3. Variables

Declaration and initialization

Variables are **containers** for storing data values.
Python has **no command for declaring a variable**.
A variable **is created the moment you first assign a value to it**.

Rules for Python variables:

- ❖A variable name must start with a letter or the underscore character
- ❖A variable name cannot start with a number
- ❖A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- ❖Variable names are case-sensitive (age, Age and AGE are three different variables)

Variables do not need to be declared with any particular type and **can even change type after they have been set**.

String variables can be declared either by using **single or double quotes.**

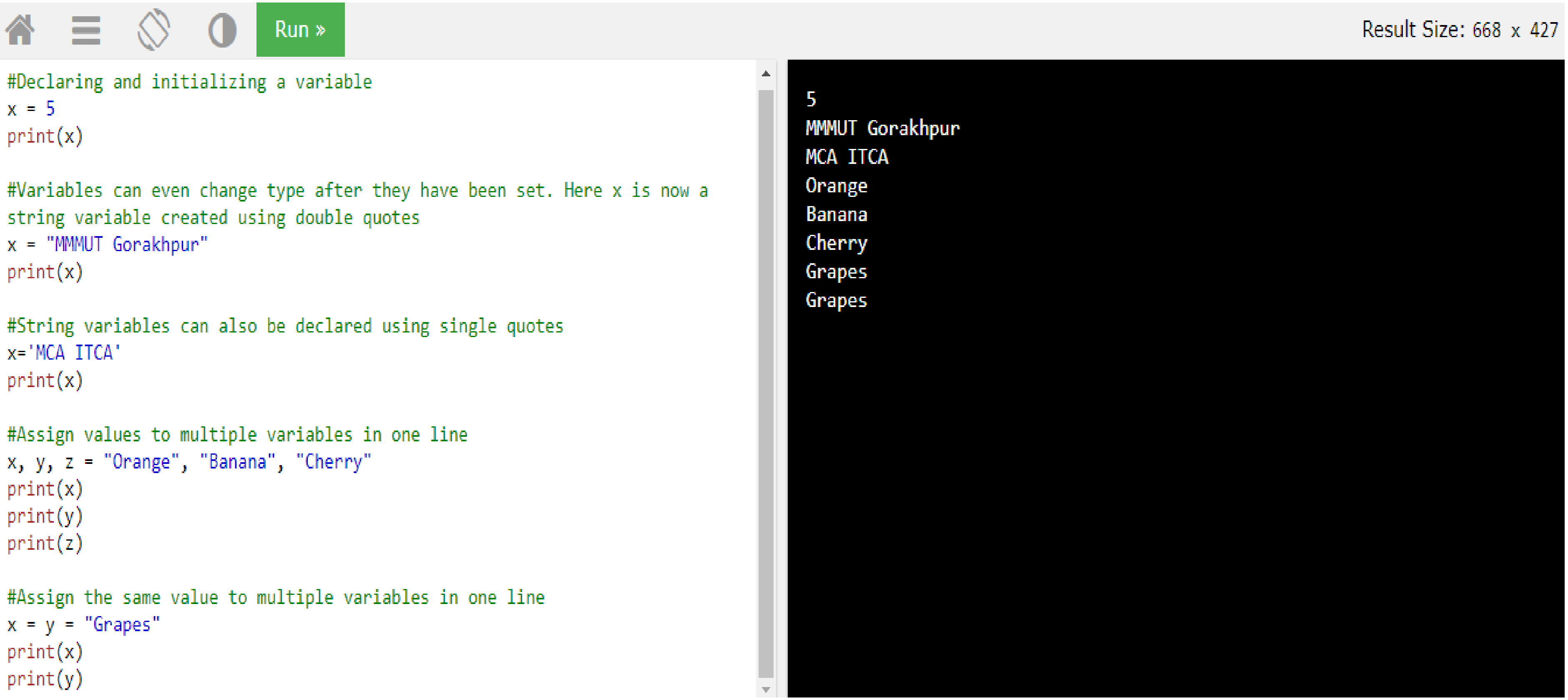
We can assign values to multiple variables in one line.

We can assign the same value to multiple variables in one line.

3. Variables

Declaration and initialization

…continued



```
#Declaring and initializing a variable
x = 5
print(x)

#Variables can even change type after they have been set. Here x is now a
string variable created using double quotes
x = "MMMUT Gorakhpur"
print(x)

#String variables can also be declared using single quotes
x='MCA ITCA'
print(x)

#Assign values to multiple variables in one line
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)

#Assign the same value to multiple variables in one line
x = y = "Grapes"
print(x)
print(y)
```

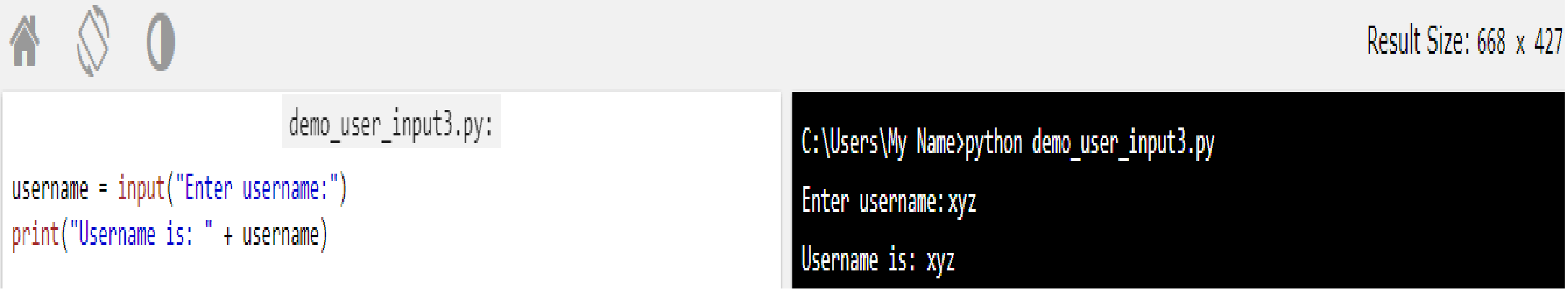
```
5
MMMUT Gorakhpur
MCA ITCA
Orange
Banana
Cherry
Grapes
Grapes
```

4. Simple Statements

4. Simple Statements

Taking inputs from user

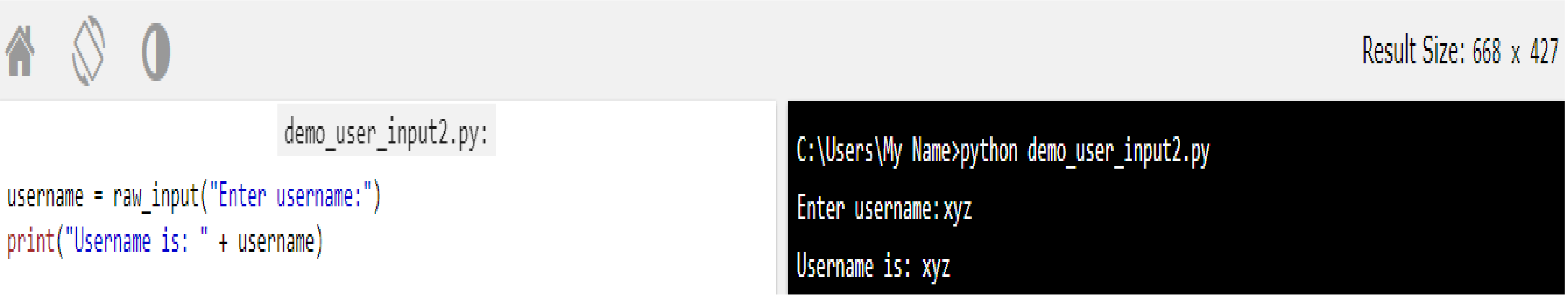
Python 3.6 uses the input( ) method while Python 2.7 uses the raw_input( ) method.



demo_user_input3.py:

```
username = input("Enter username:")
print("Username is: " + username)
```

```
C:\Users\My Name>python demo_user_input3.py
Enter username:xyz
Username is: xyz
```



demo_user_input2.py:

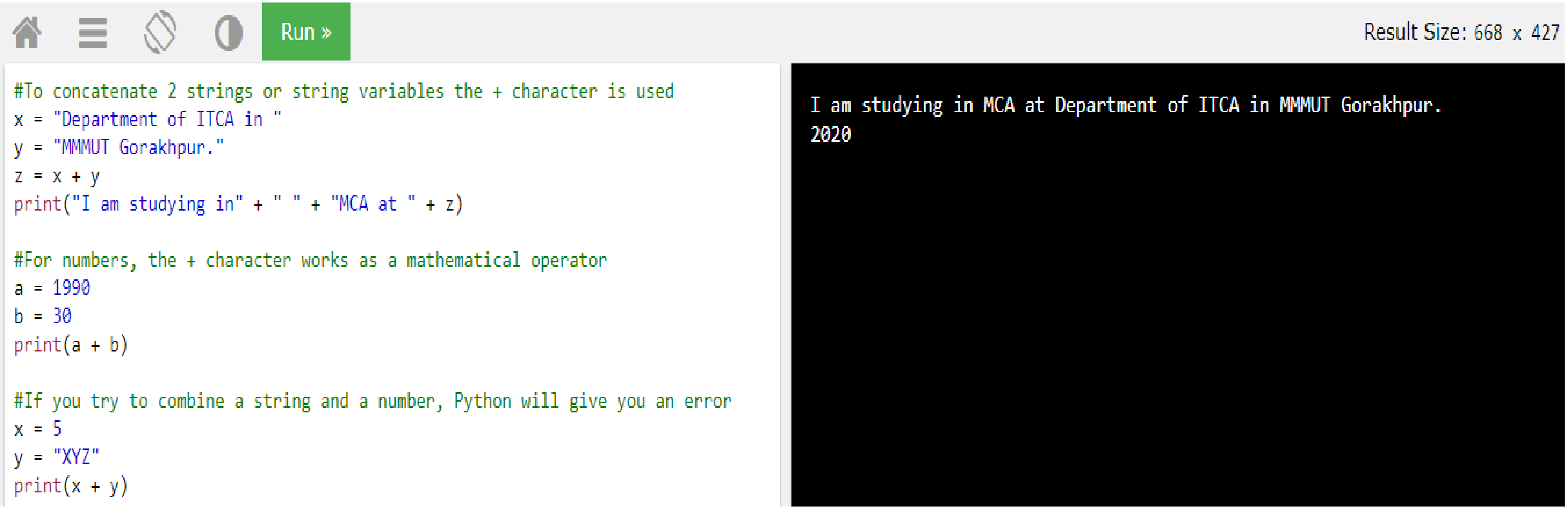
```
username = raw_input("Enter username:")
print("Username is: " + username)
```

```
C:\Users\My Name>python demo_user_input2.py
Enter username:xyz
Username is: xyz
```

4. Simple Statements

Displaying outputs

print statement is often used to output variables.
To concatenate two or more strings or string variables the + character is used.
For numbers, the + character works as a mathematical operator.
If you try to combine a string and a number, Python will give you an error.



```
#To concatenate 2 strings or string variables the + character is used
x = "Department of ITCA in "
y = "MMMUT Gorakhpur."
z = x + y
print("I am studying in" + " " + "MCA at " + z)

#For numbers, the + character works as a mathematical operator
a = 1990
b = 30
print(a + b)

#If you try to combine a string and a number, Python will give you an error
x = 5
y = "XYZ"
print(x + y)
```

```
I am studying in MCA at Department of ITCA in MMMUT Gorakhpur.
2020
```

5. Other Concepts

5. Other Concepts

Operators and Expressions

- Arithmetic operators (+, -, *, /, %, **, //)
- Assignment operators (=, +=, -=, *=, /=, %=, **=, //=, &=, |=, <<=, >>=, ^=)
- Comparison operators (==, !=, <, >, <=, >=)
- Logical operators (and, or, not)
- Identity operators (is, is not)
- Membership operators (in, not in)
- Bitwise operators (&, |, ^, ~, <<, >>)

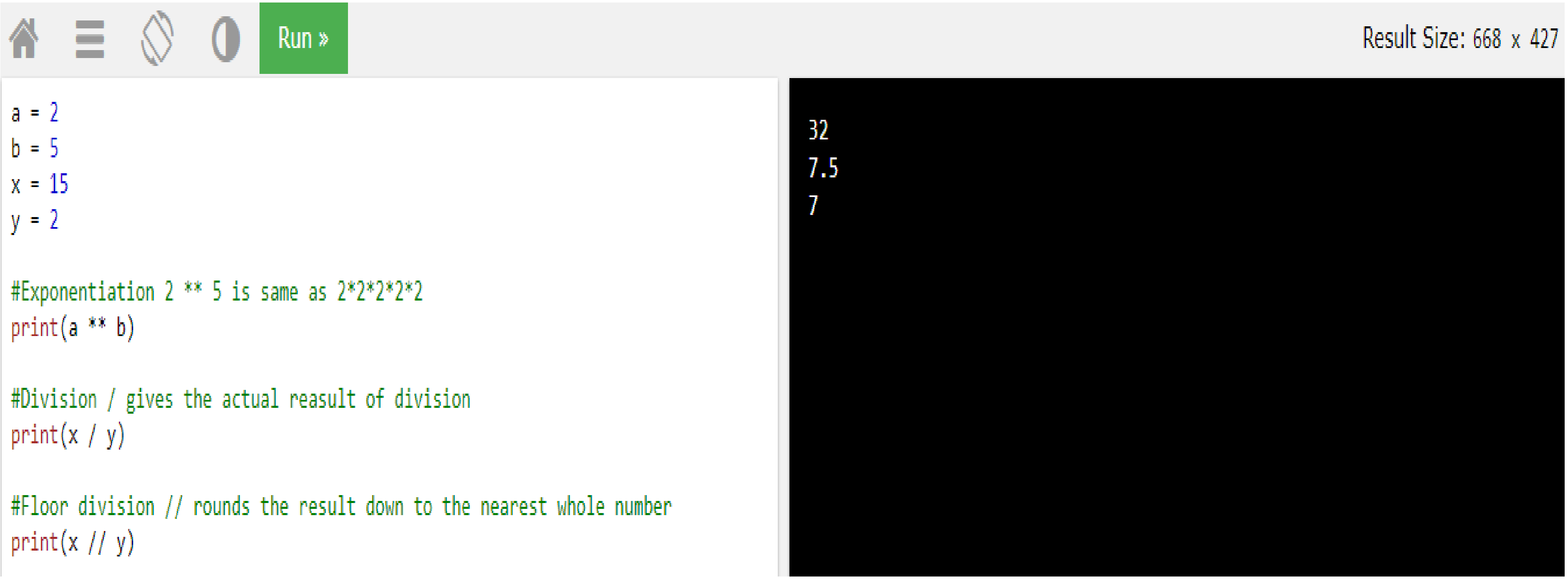
5. Other Concepts

Operators and Expressions

…continued

Arithmetic operators

** is used for Exponentiation and // is used for Floor division.



```
a = 2
b = 5
x = 15
y = 2

#Exponentiation 2 ** 5 is same as 2*2*2*2*2
print(a ** b)

#Division / gives the actual reasult of division
print(x / y)

#Floor division // rounds the result down to the nearest whole number
print(x // y)
```

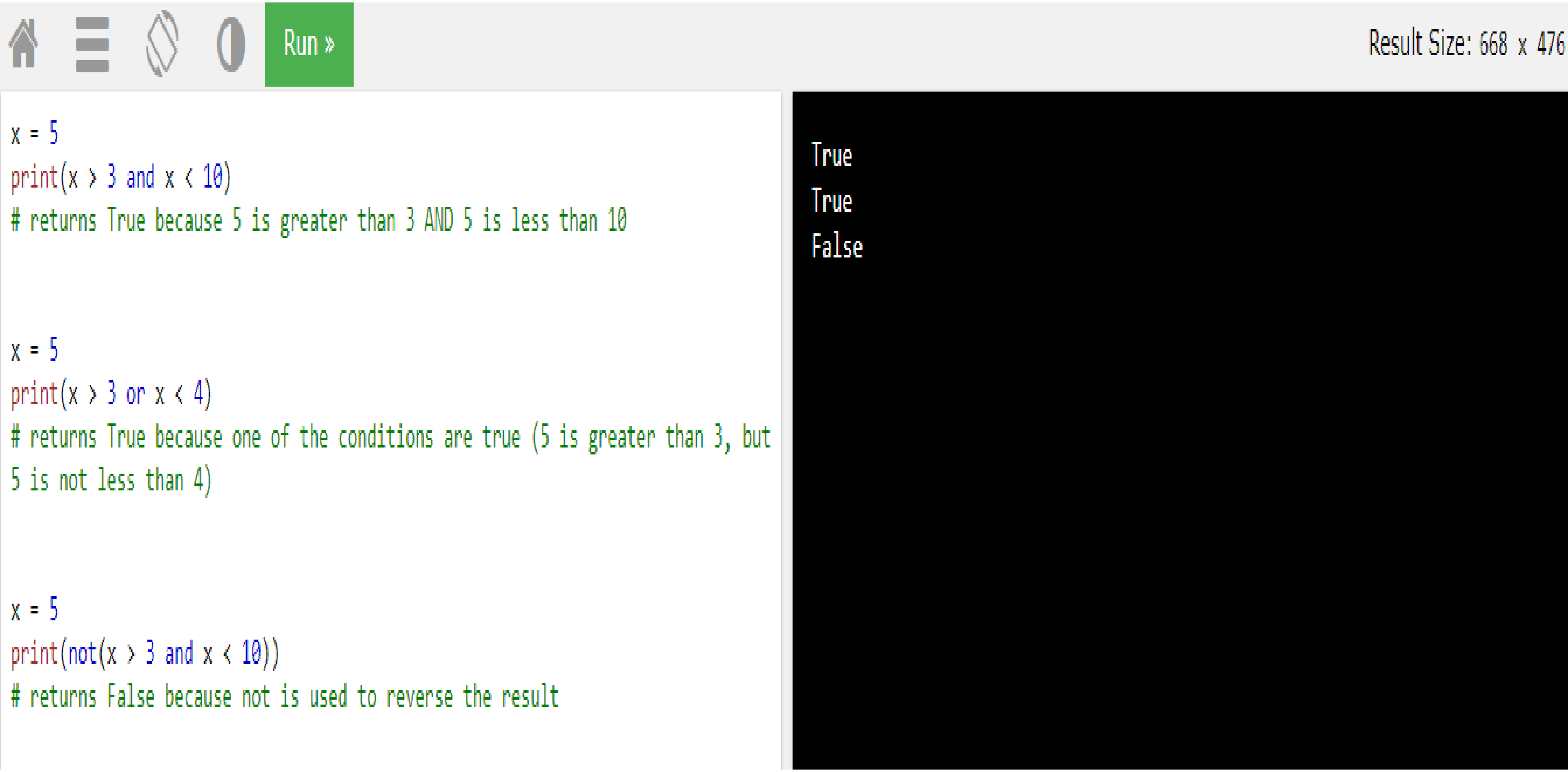
```
32
7.5
7
```

5. Other Concepts

Operators and Expressions

…continued

Logical operators



```
x = 5
print(x > 3 and x < 10)
# returns True because 5 is greater than 3 AND 5 is less than 10


x = 5
print(x > 3 or x < 4)
# returns True because one of the conditions are true (5 is greater than 3, but
5 is not less than 4)


x = 5
print(not(x > 3 and x < 10))
# returns False because not is used to reverse the result
```

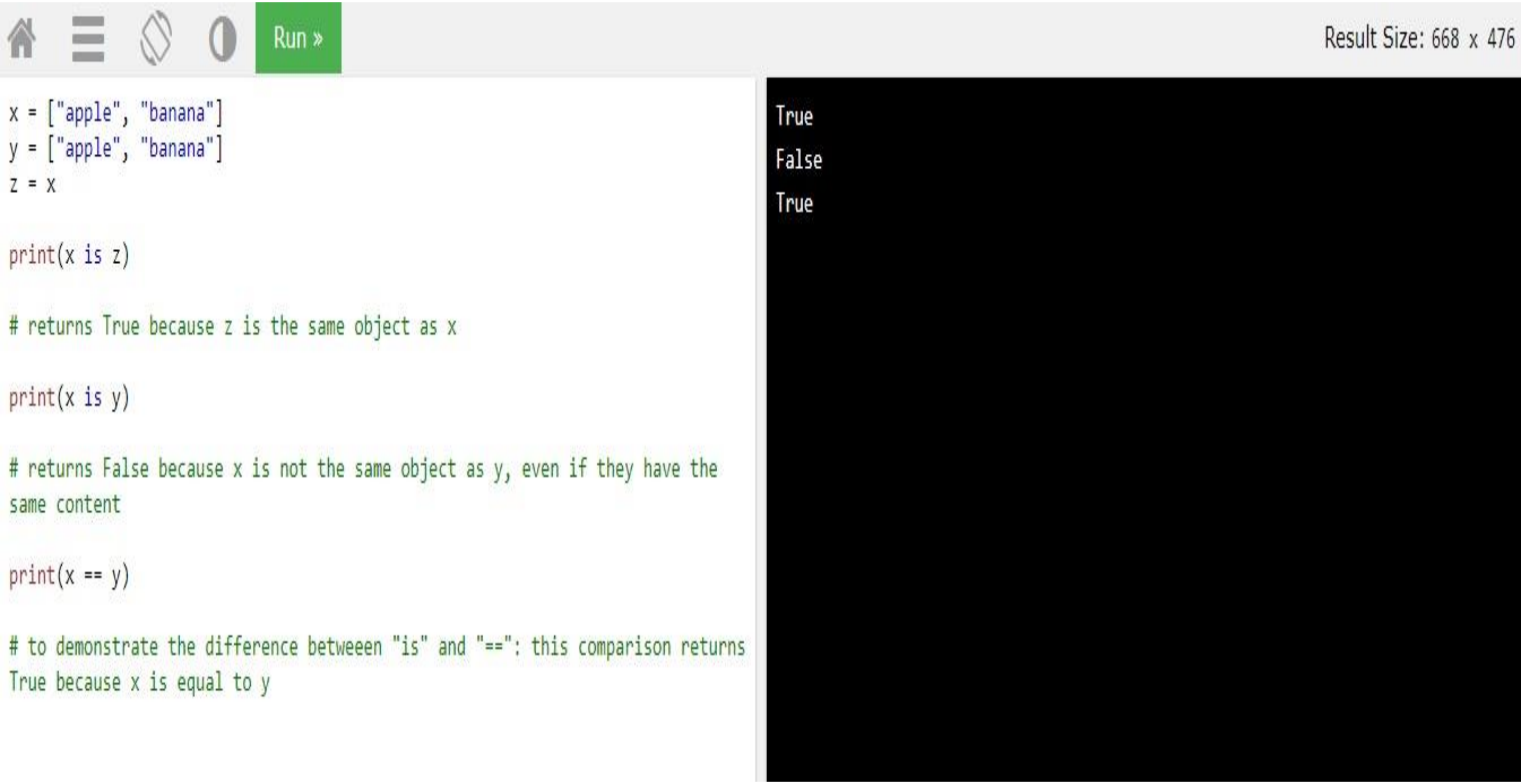
```
True
True
False
```

5. Other Concepts

Operators and Expressions

…continued

Identity operators



```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is z)

# returns True because z is the same object as x

print(x is y)

# returns False because x is not the same object as y, even if they have the
same content

print(x == y)

# to demonstrate the difference betweeen "is" and "==": this comparison returns
True because x is equal to y
```

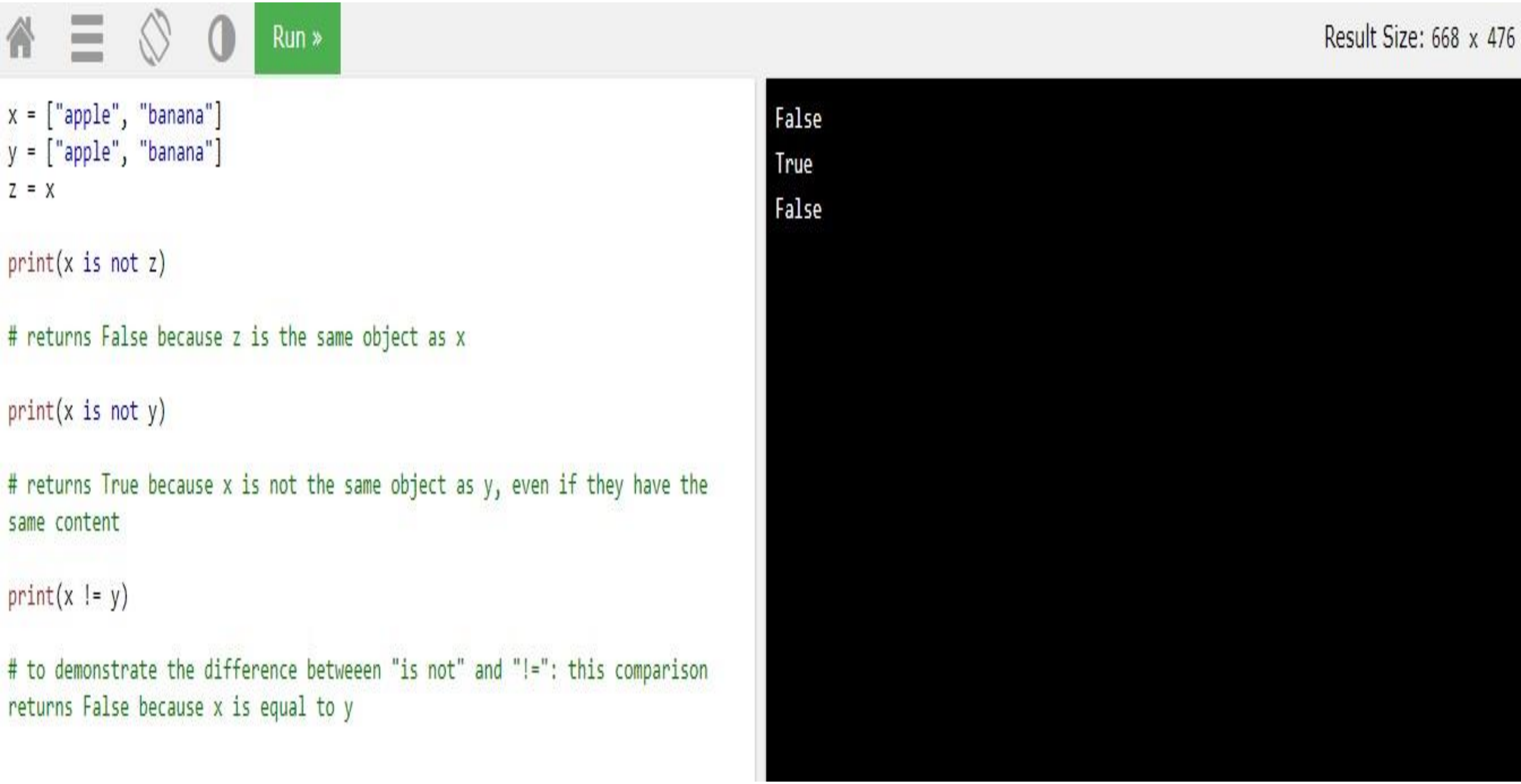
```
True
False
True
```

5. Other Concepts

Operators and Expressions

…continued

Identity operators



```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is not z)

# returns False because z is the same object as x

print(x is not y)

# returns True because x is not the same object as y, even if they have the
same content

print(x != y)

# to demonstrate the difference betweeen "is not" and "!=": this comparison
returns False because x is equal to y
```

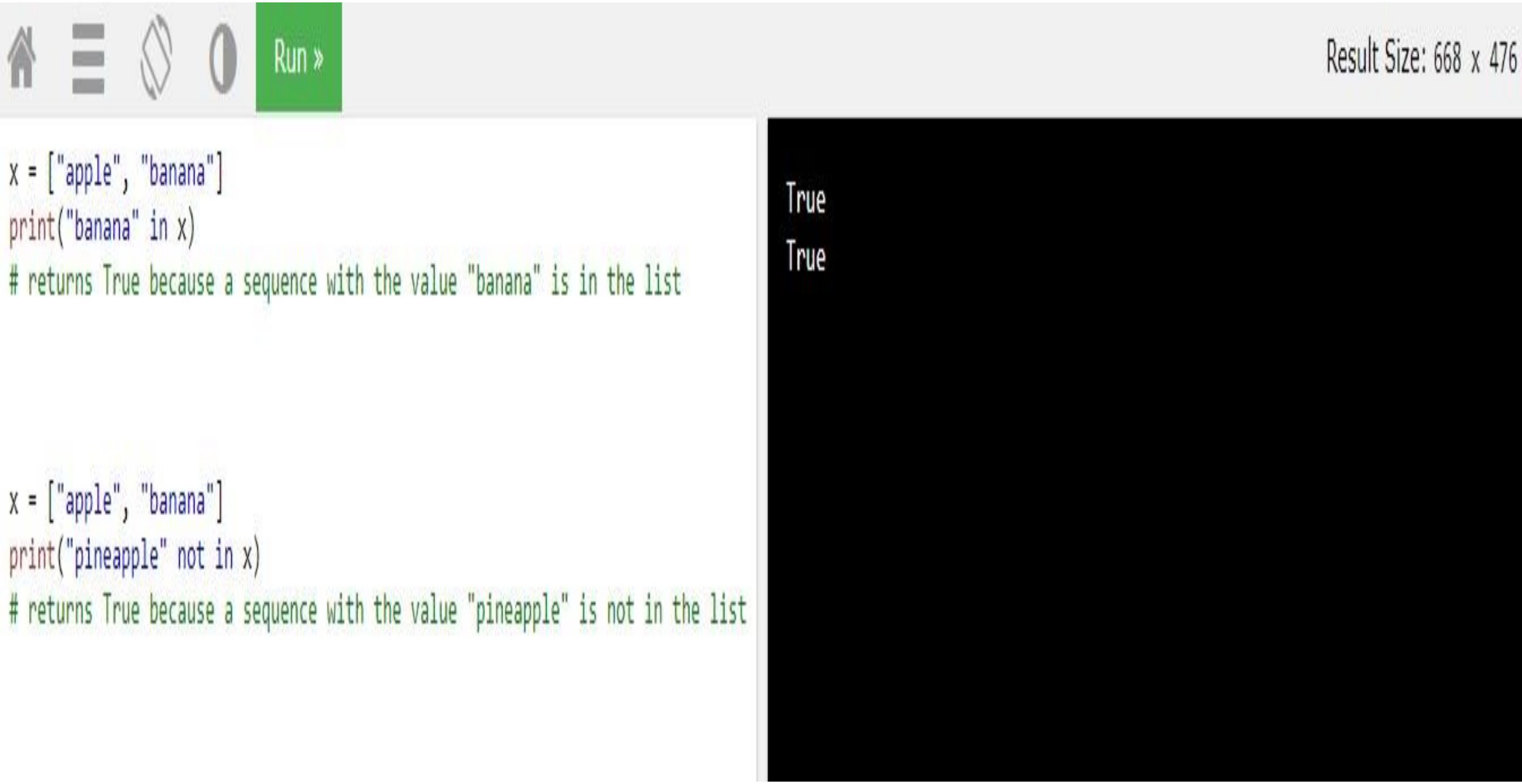
```
False
True
False
```

5. Other Concepts

Operators and Expressions

…continued

Membership operators



```
x = ["apple", "banana"]
print("banana" in x)
# returns True because a sequence with the value "banana" is in the list



x = ["apple", "banana"]
print("pineapple" not in x)
# returns True because a sequence with the value "pineapple" is not in the list
```

```
True
True
```

5. Other Concepts

Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code. Other languages often use curly-brackets for this purpose.

Python will give you an error if you skip the indentation.

The number of spaces is up to you as a programmer, but it has to be at least one.

You have to use same number of spaces in the same block of code, otherwise Python will give you an error.

Correct Example:

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

5. Other Concepts

Comments

Comments starts with a #, and Python will ignore them.

Python does not really have a syntax for multi line comments.

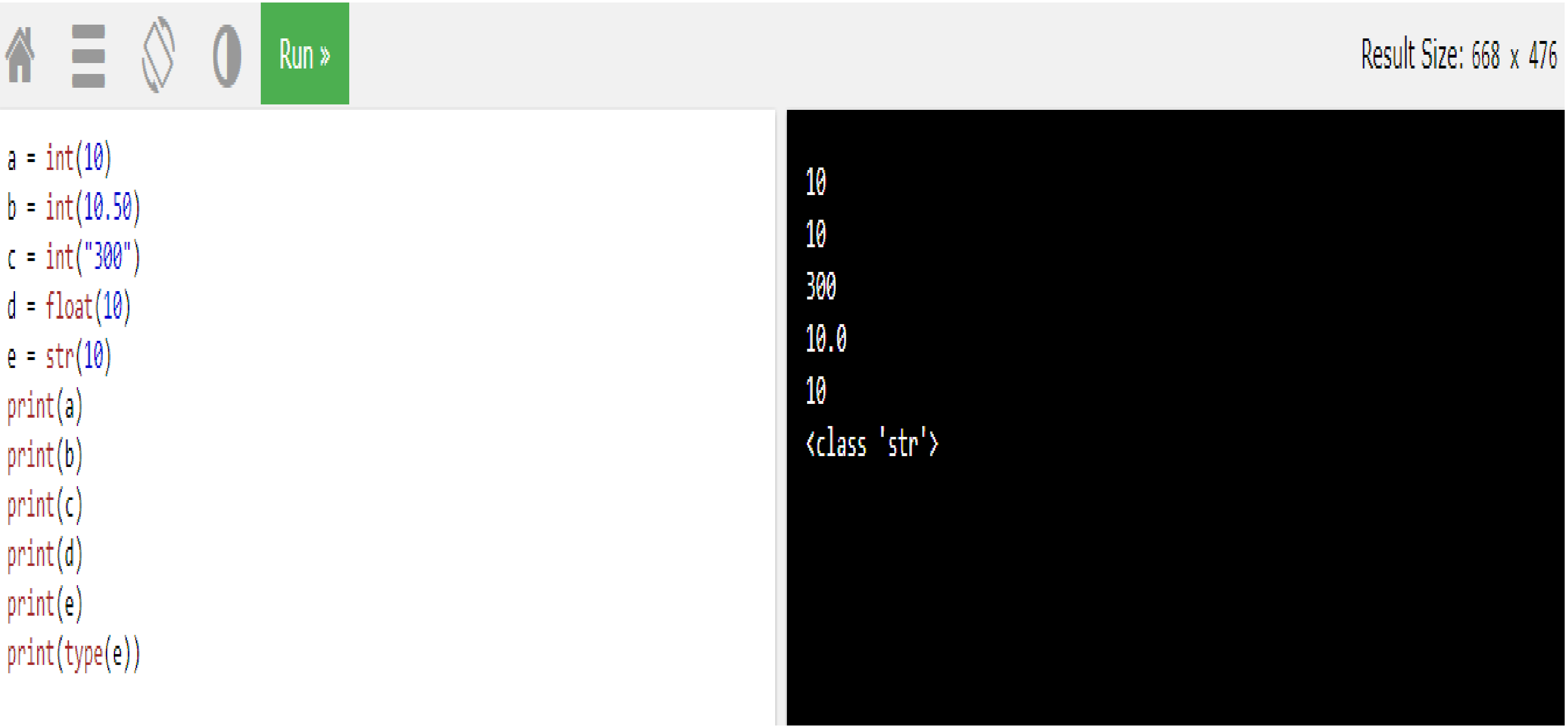
To add a multiline comment you could insert a # for each line.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it.

5. Other Concepts

Casting

Casting can be done using constructors : int( ), float( ), str( )



```
a = int(10)
b = int(10.50)
c = int("300")
d = float(10)
e = str(10)
print(a)
print(b)
print(c)
print(d)
print(e)
print(type(e))
```

```
10
10
300
10.0
10
<class 'str'>
```

6. Conditional Statements

6. Conditional Statements

if…elif…else

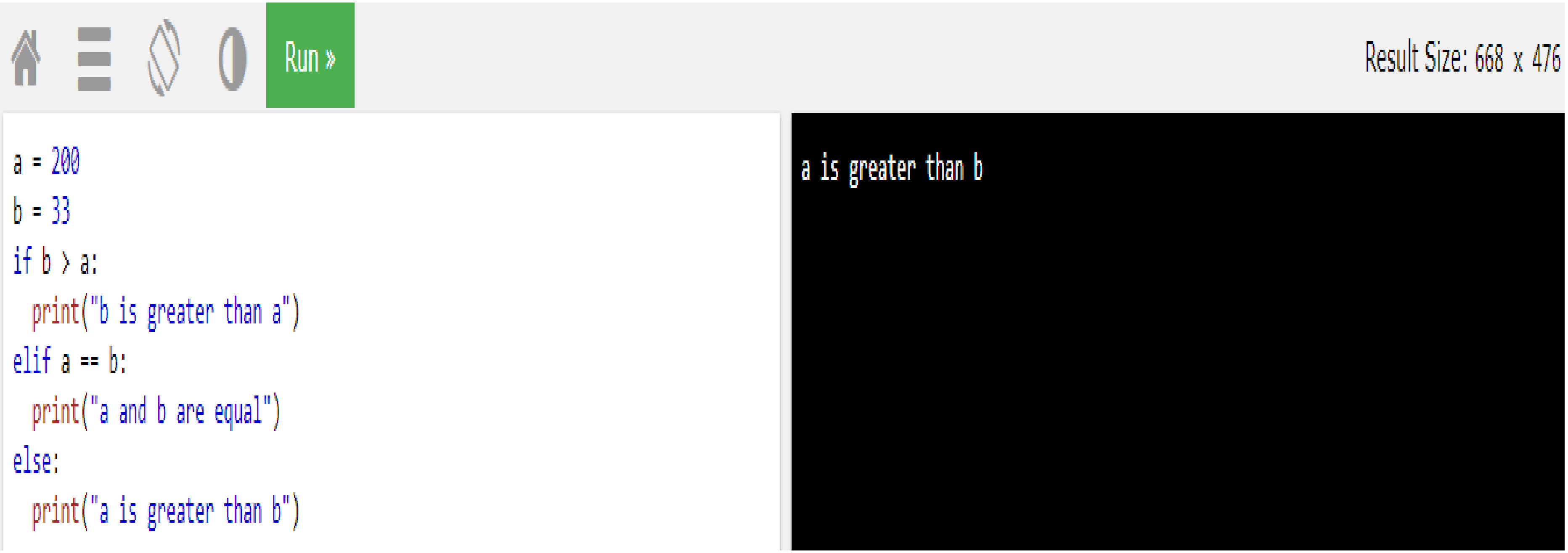
```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

If you have only one statement to execute, you can put it on the same line as the if statement.

```
if a > b: print("a is greater than b")
```

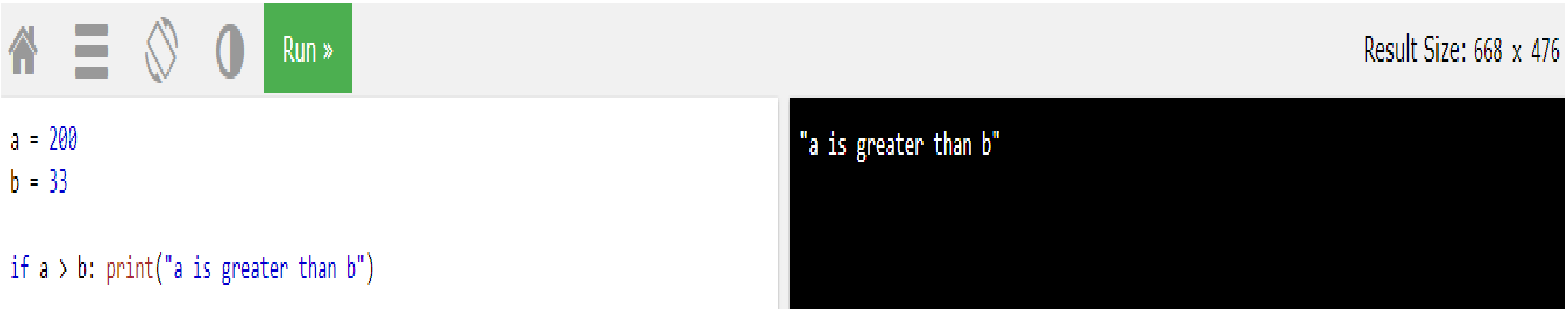
6. Conditional Statements

if…elif…else



```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a is greater than b
```



```
a = 200
b = 33

if a > b: print("a is greater than b")
```

```
"a is greater than b"
```

6. Conditional Statements

if…elif…else

Ternary Operators, or **Conditional Expressions**

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line.

```
a = 2
b = 330
print("A") if a > b else print("B")
```

You can also have multiple else statements on the same line.

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

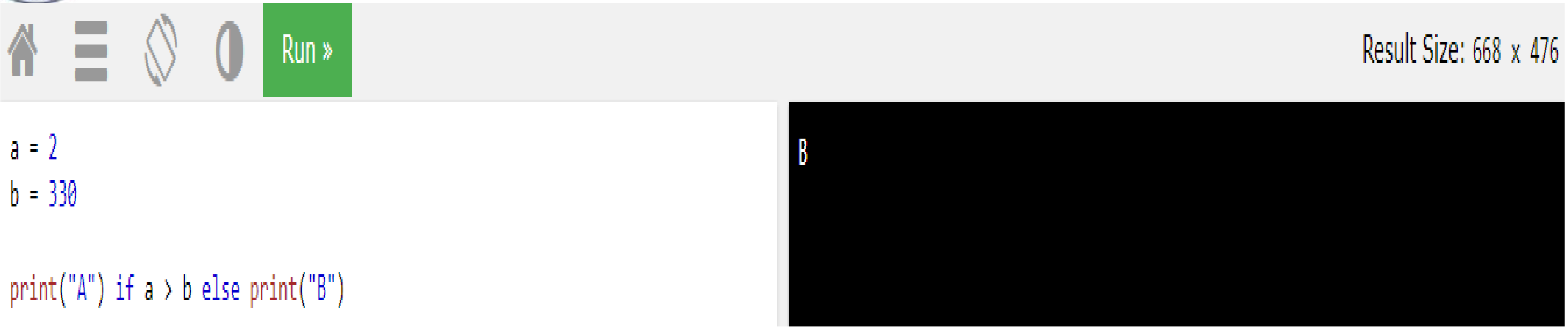
Logical operators to combine conditional statements

The and/or keywords are logical operators, and can be used as below:

```
if a > b and a > c:

if a > b or a > c:
```

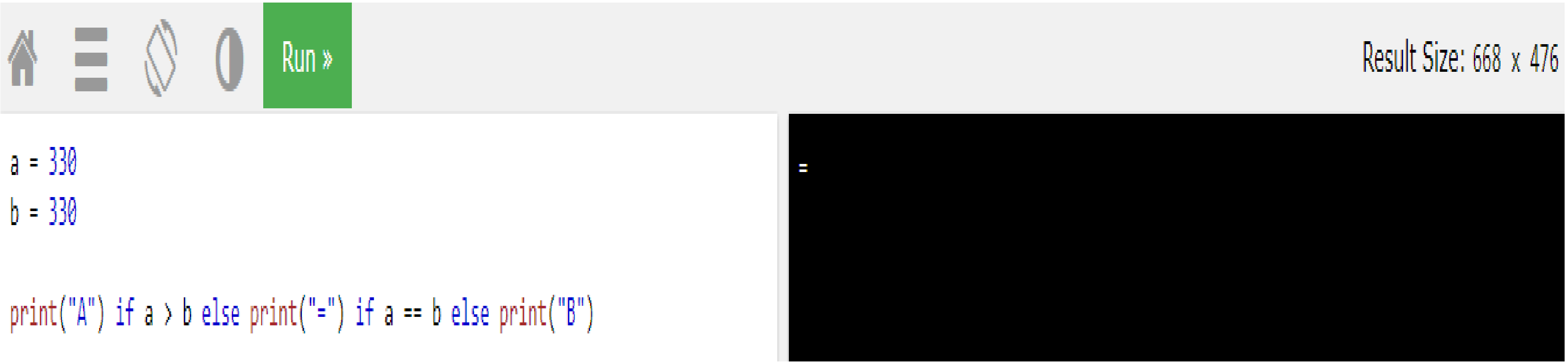
6. Conditional Statements

Ternary operator



```python
a = 2
b = 330

print("A") if a > b else print("B")
```

```
B
```

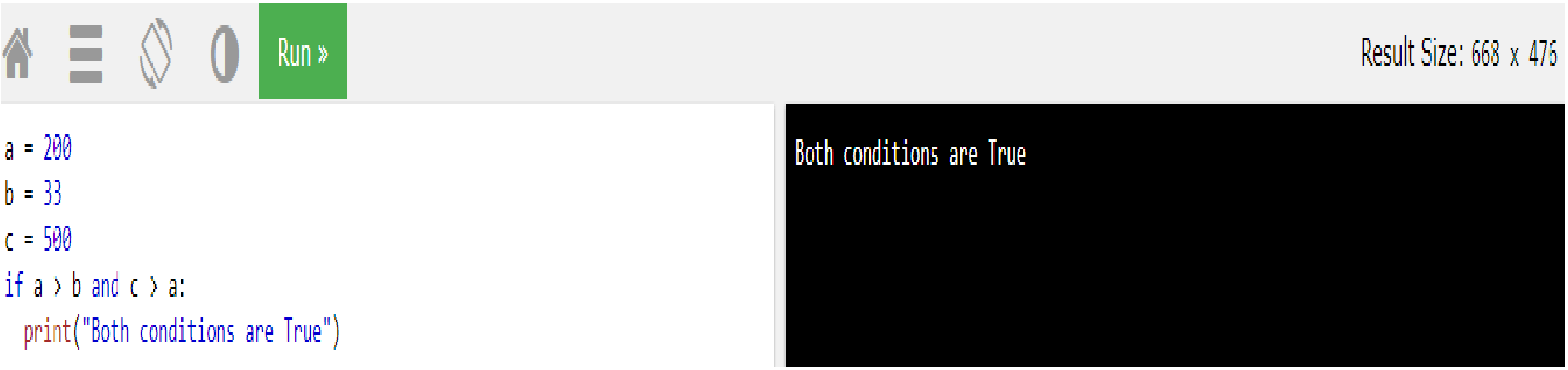


```python
a = 330
b = 330

print("A") if a > b else print("=") if a == b else print("B")
```

```
=
```

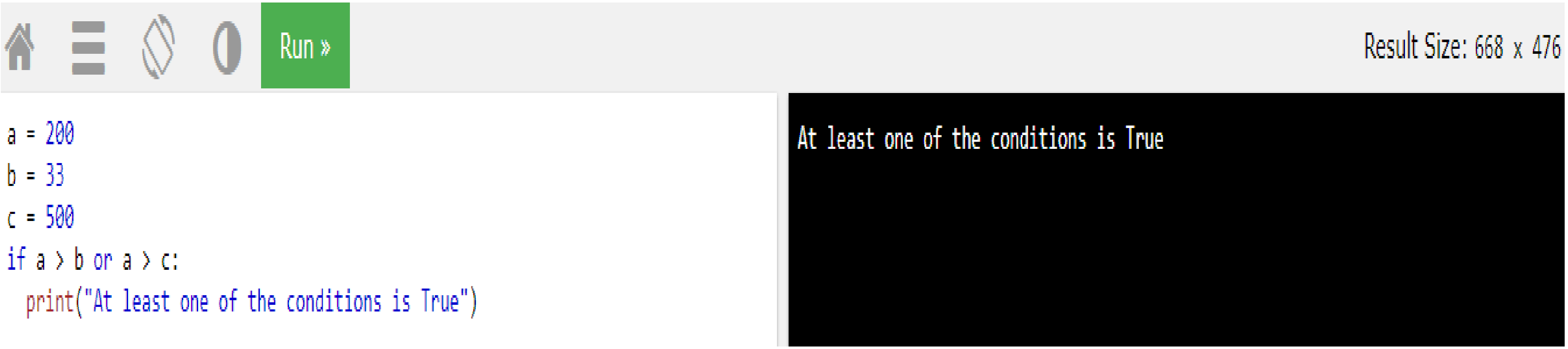
6. Conditional Statements

Logical operator



```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

```
Both conditions are True
```



```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

```
At least one of the conditions is True
```

6. Conditional Statements

if…elif…else

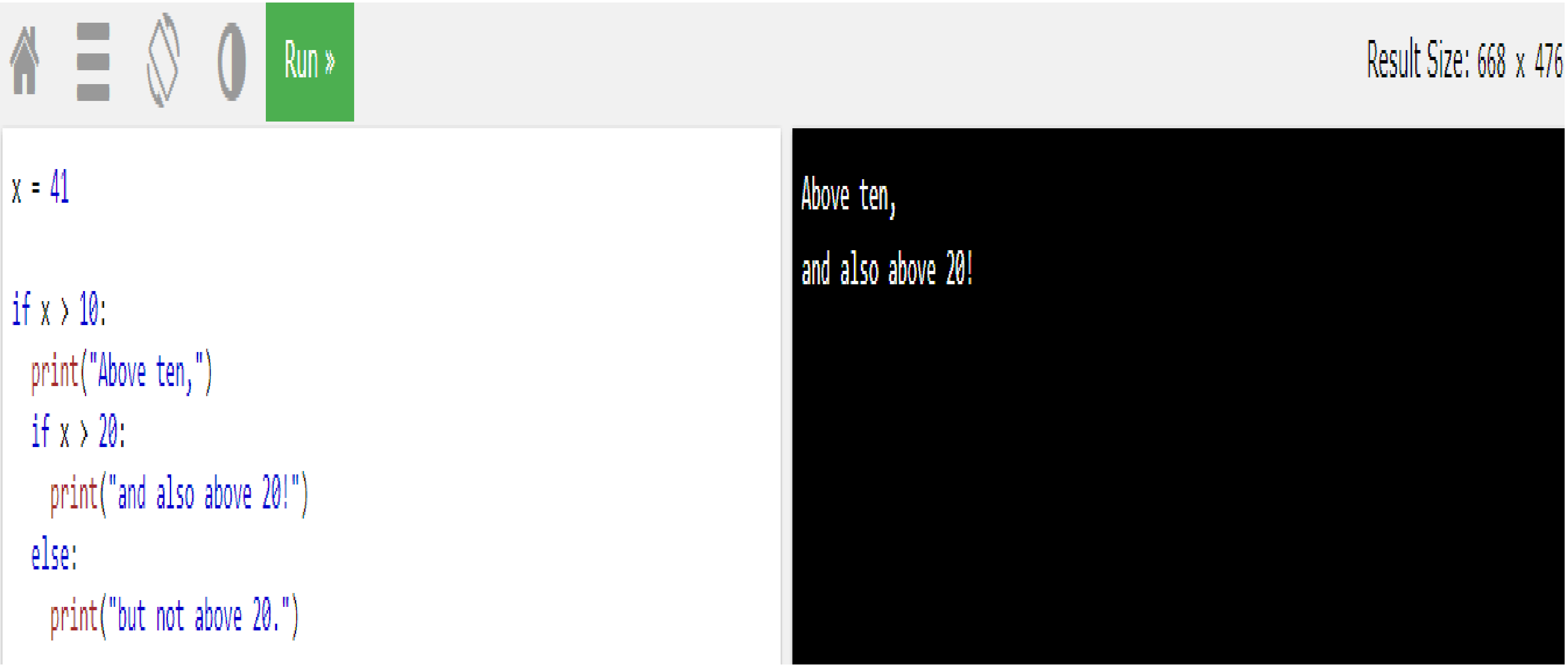
Nested if

You can have if statements inside if statements, this is called nested if statements.

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

6. Conditional Statements

Nested if



```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

```
Above ten,
and also above 20!
```

6. Conditional Statements

if…elif…else

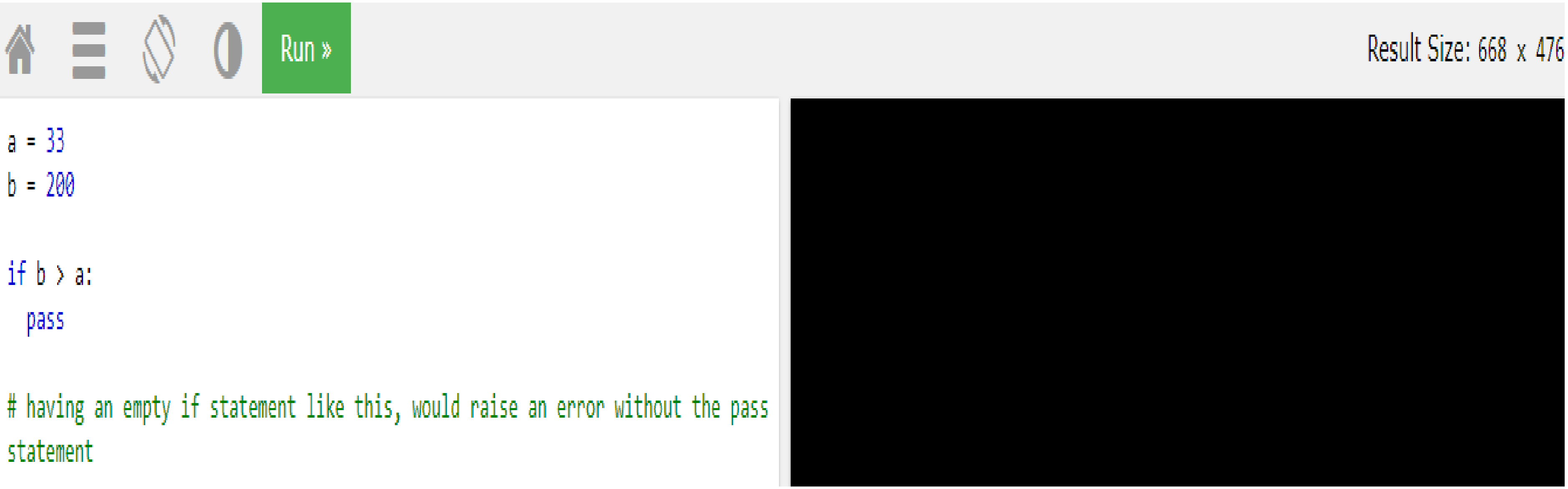
Pass statement

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

```
a = 33
b = 200
if b > a:
  pass
```

6. Conditional Statements

Pass statement



```
a = 33
b = 200

if b > a:
  pass

# having an empty if statement like this, would raise an error without the pass statement
```

Queries ?