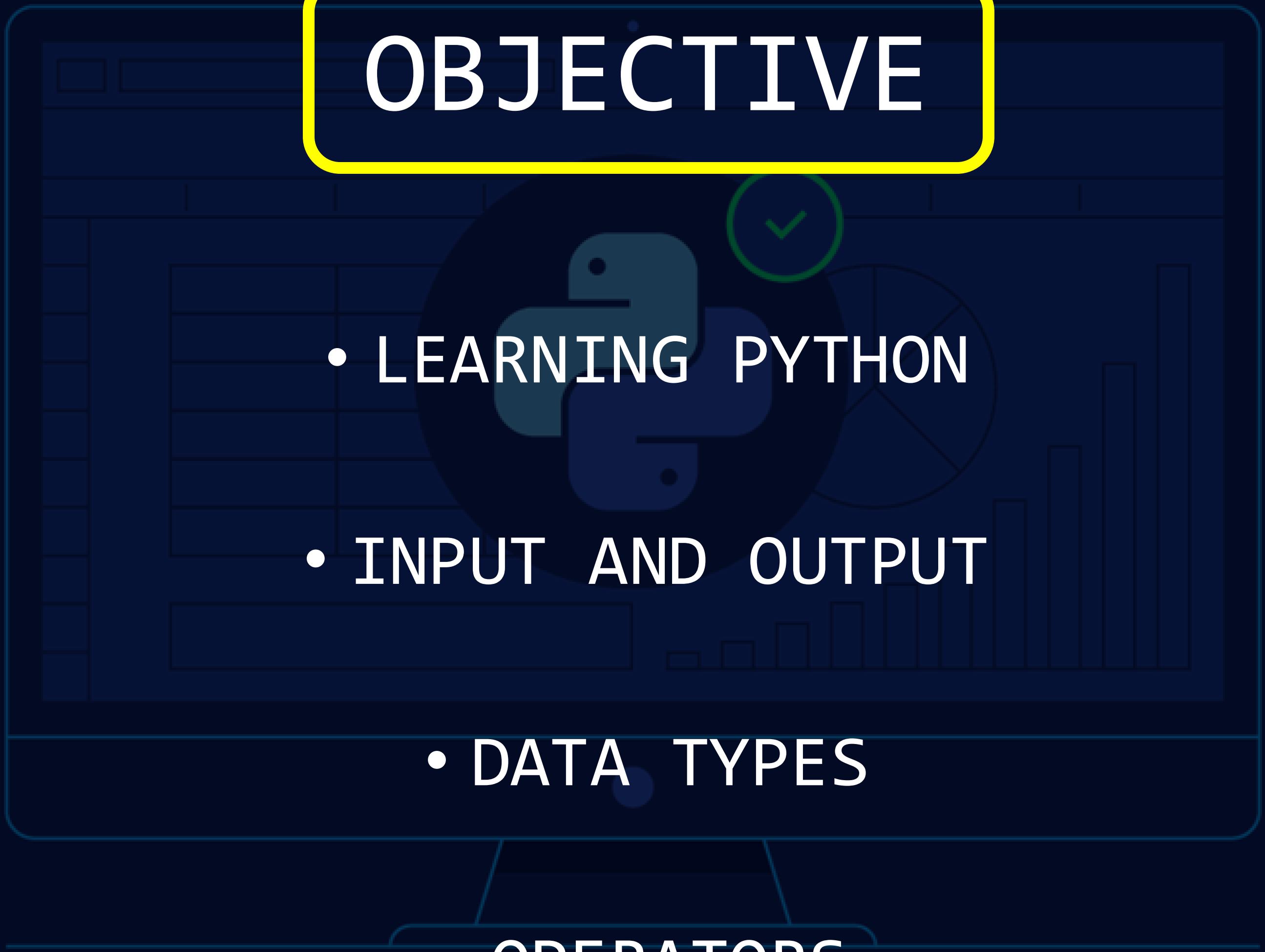
Learning python

20220928

4110E234

OBJECTIVE

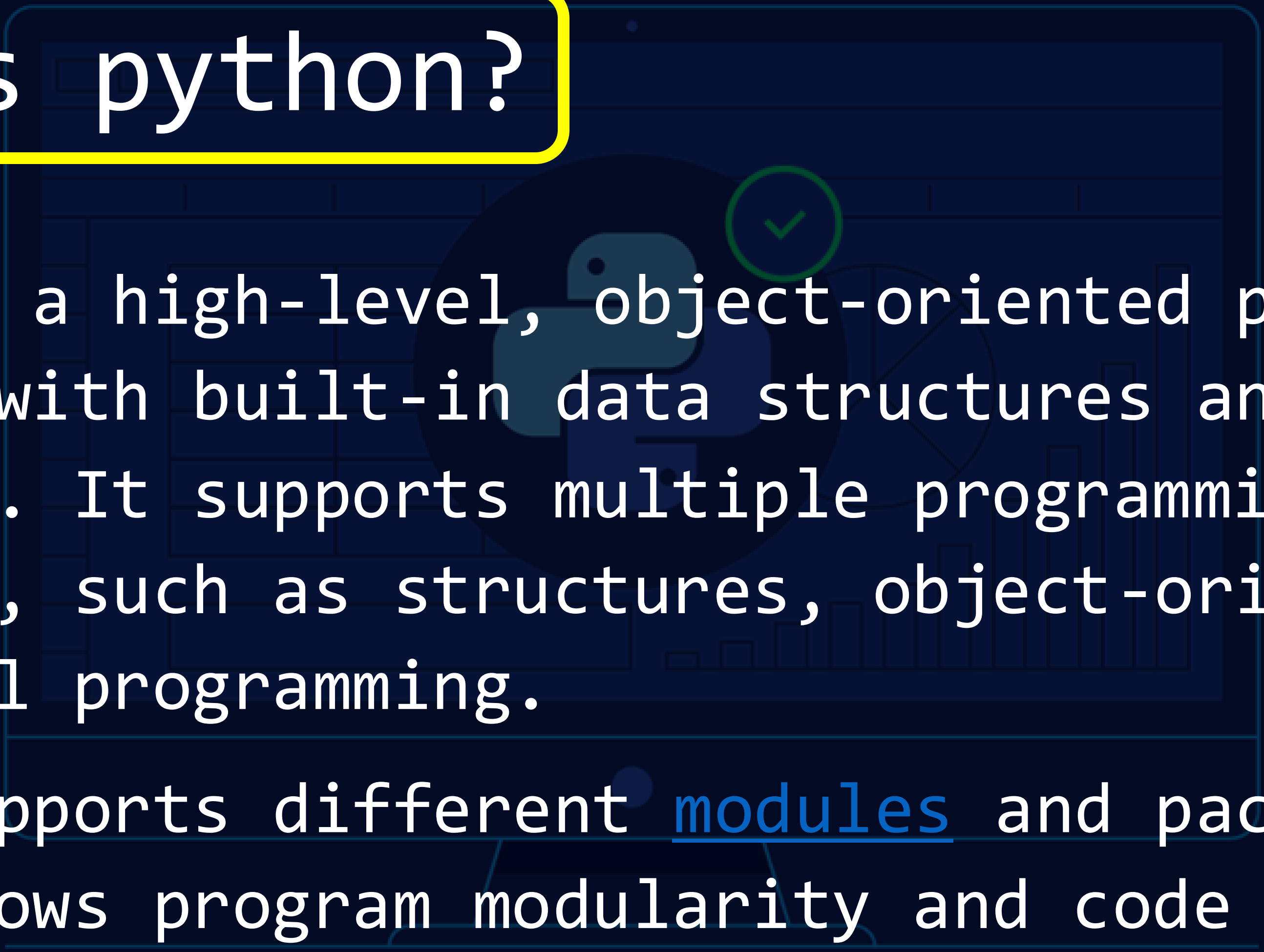


- LEARNING PYTHON
- INPUT AND OUTPUT
- DATA TYPES
- OPERATORS

CONTENT

- What is Python?
- Python Jobs
- Why to Learn Python?
- Python Online Interpreter
    - One
    - Two
- Google Colab
- Python Codes

What is python?



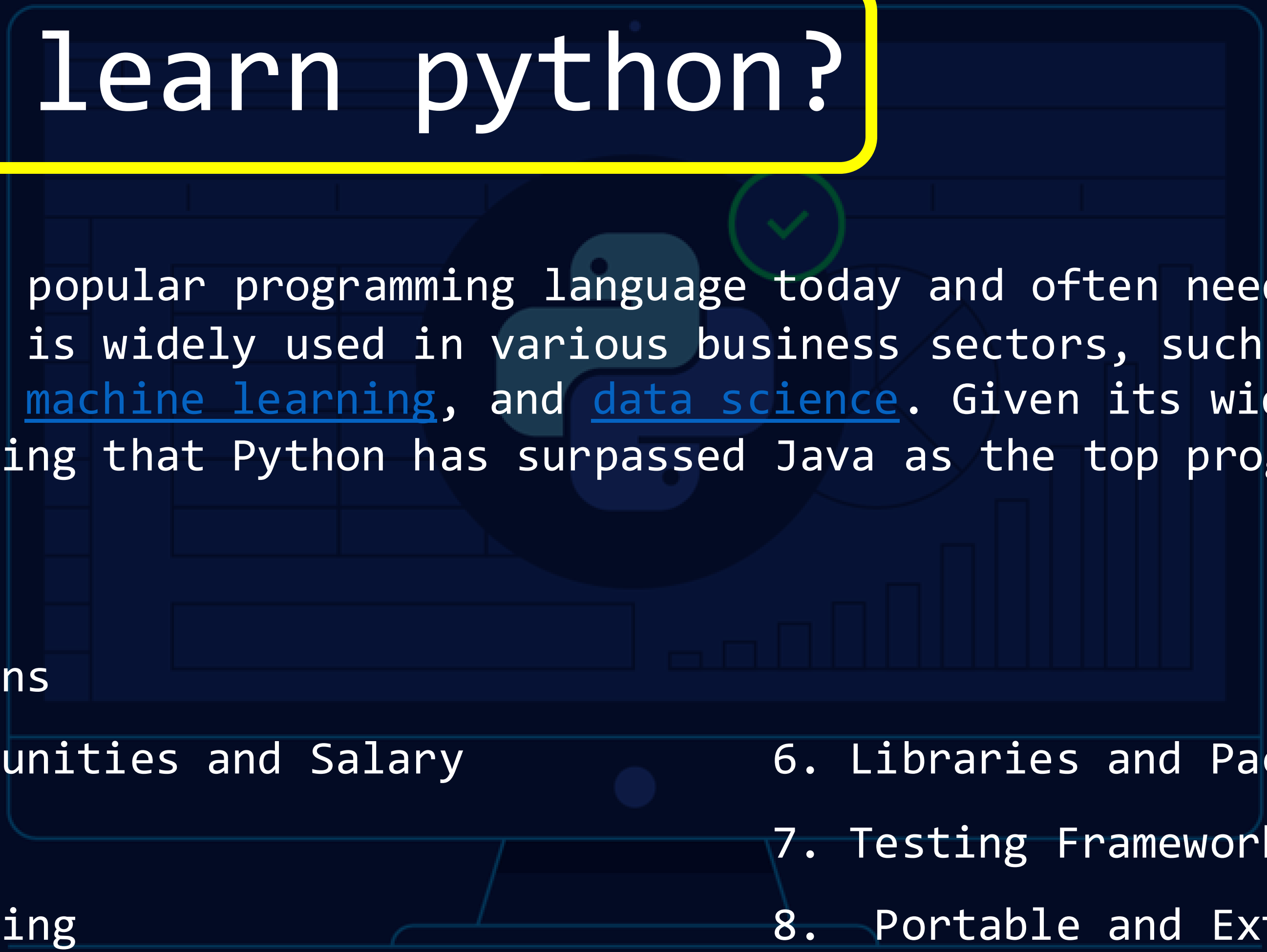
- Python is a high-level, object-oriented programming language with built-in data structures and dynamic semantics. It supports multiple programming paradigms, such as structures, object-oriented, and functional programming.
- Python supports different modules and packages, which allows program modularity and code reuse.
- Python was created by Guido van Rossum.

Python Jobs

- Career Opportunities

Python language provides several job opportunities and promises high growth with huge salary prospects. Some of the big and renowned companies that use Python for their development are:

Why to learn python?



Python is a very popular programming language today and often needs an introduction. It is widely used in various business sectors, such as programming, web development, machine learning, and data science. Given its widespread use, it’s not surprising that Python has surpassed Java as the top programming language.

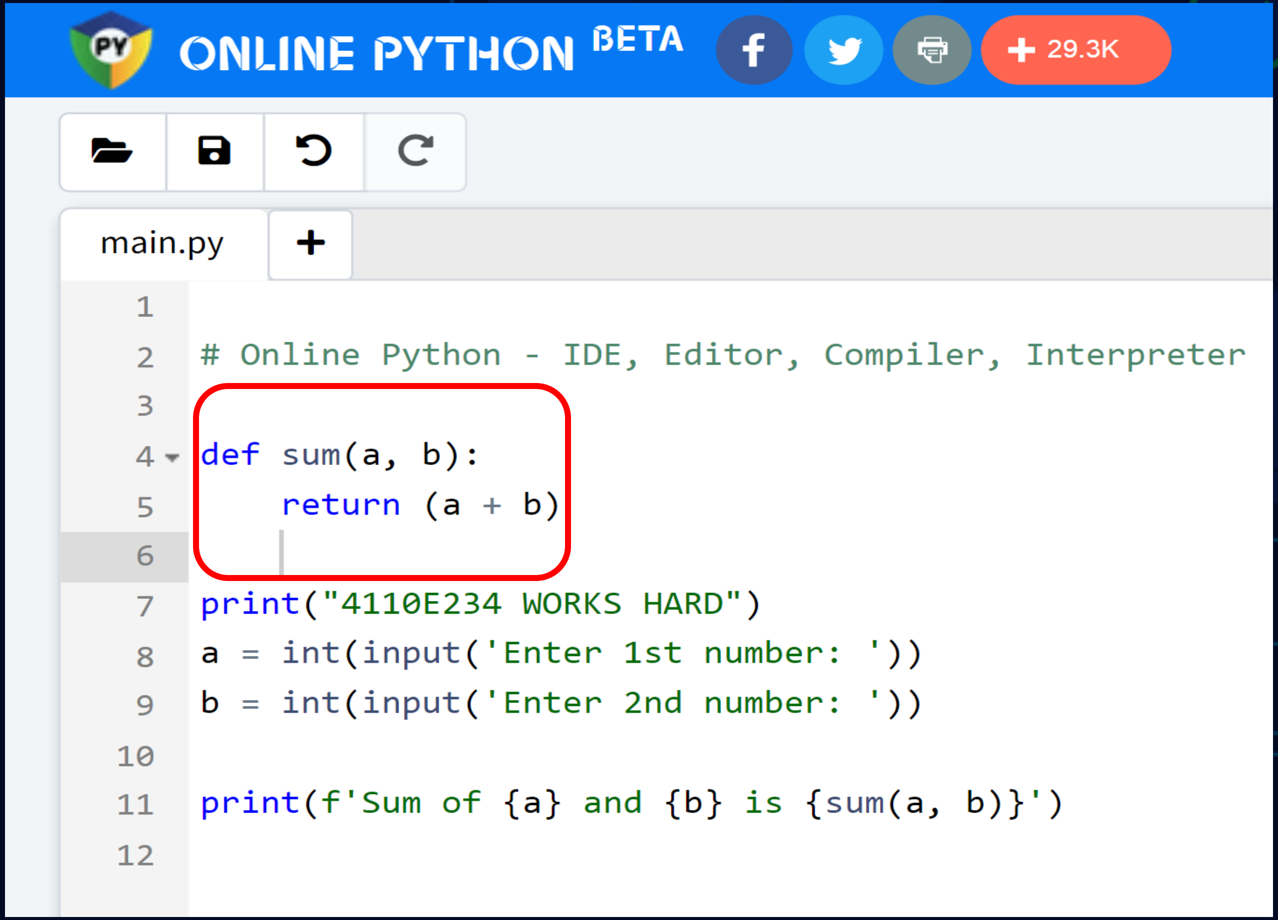
The Top 10 Reasons

1. Career Opportunities and Salary
2. Data Science
3. Machine Learning
4. Web Development
5. Scripting and Automation
6. Libraries and Packages
7. Testing Frameworks
8. Portable and Extensible
9. Active Community
10. Easy to Use

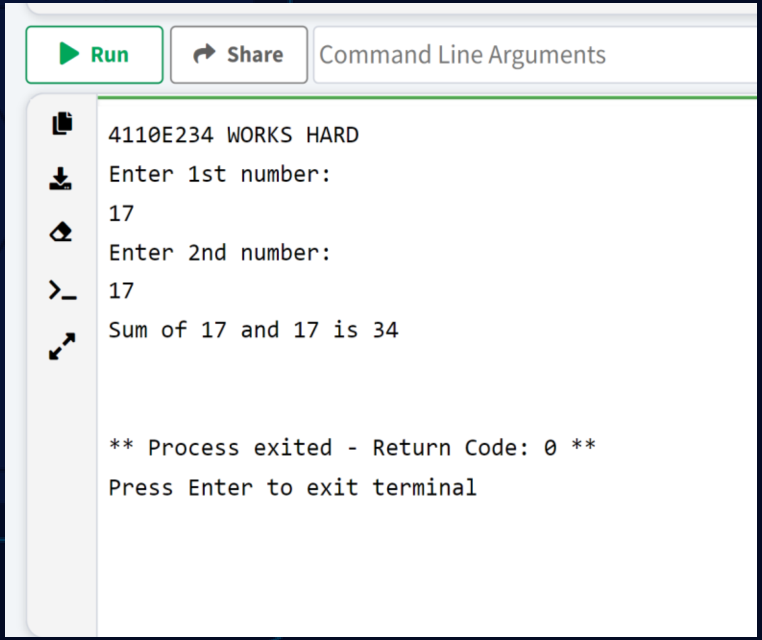
Why to learn python?

- Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Python Online Interpreter: 1

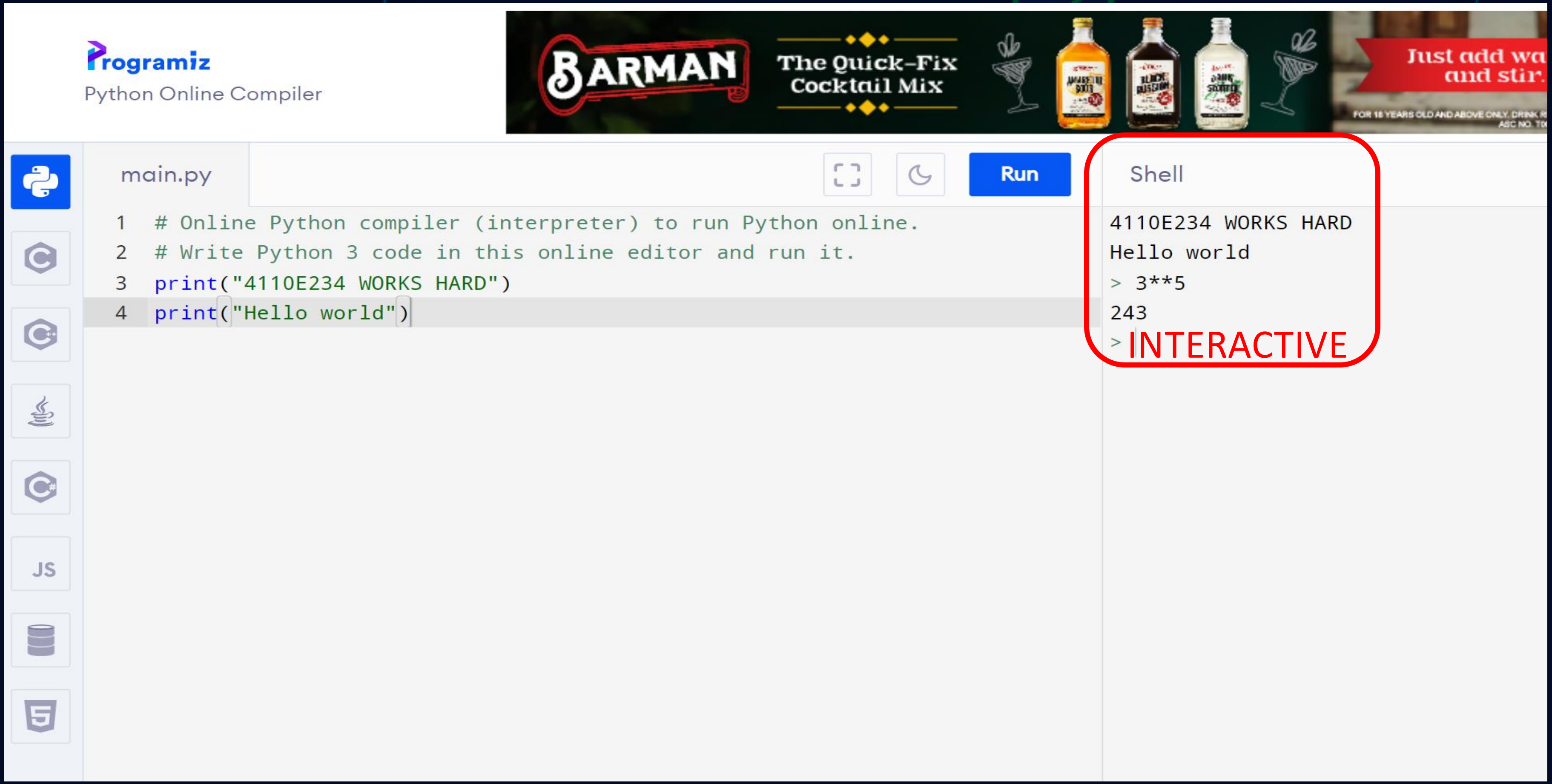


```
# Online Python - IDE, Editor, Compiler, Interpreter

def sum(a, b):
    return (a + b)

print("4110E234 WORKS HARD")
a = int(input('Enter 1st number: '))
b = int(input('Enter 2nd number: '))

print(f'Sum of {a} and {b} is {sum(a, b)}')
```



```
4110E234 WORKS HARD
Enter 1st number:
17
Enter 2nd number:
17
Sum of 17 and 17 is 34


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Python Online Interpreter: 2



```
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
print("4110E234 WORKS HARD")
print("Hello world")
```

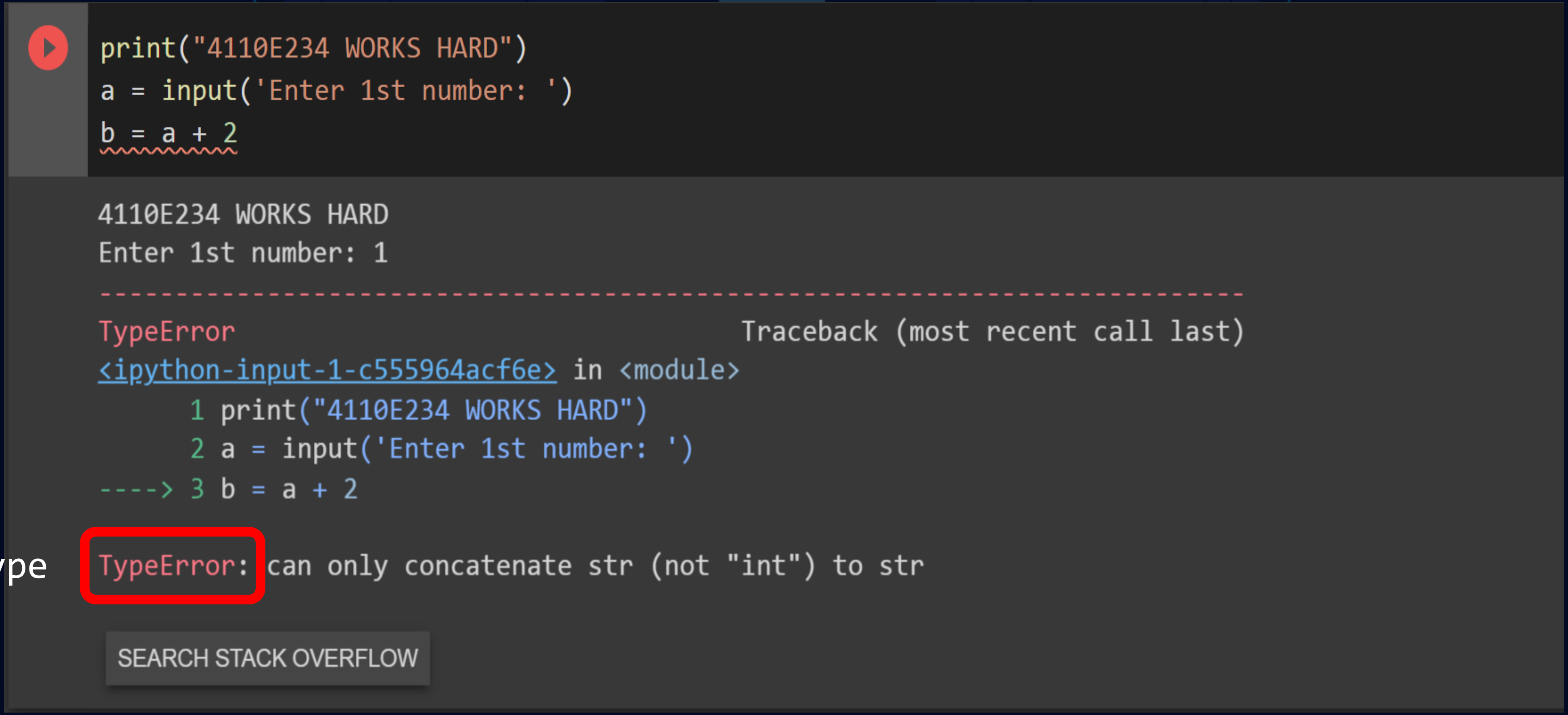
```
4110E234 WORKS HARD
Hello world
> 3**5
243
>
```

INTERACTIVE

INPUT AND OUTPUT

20220928

Question:

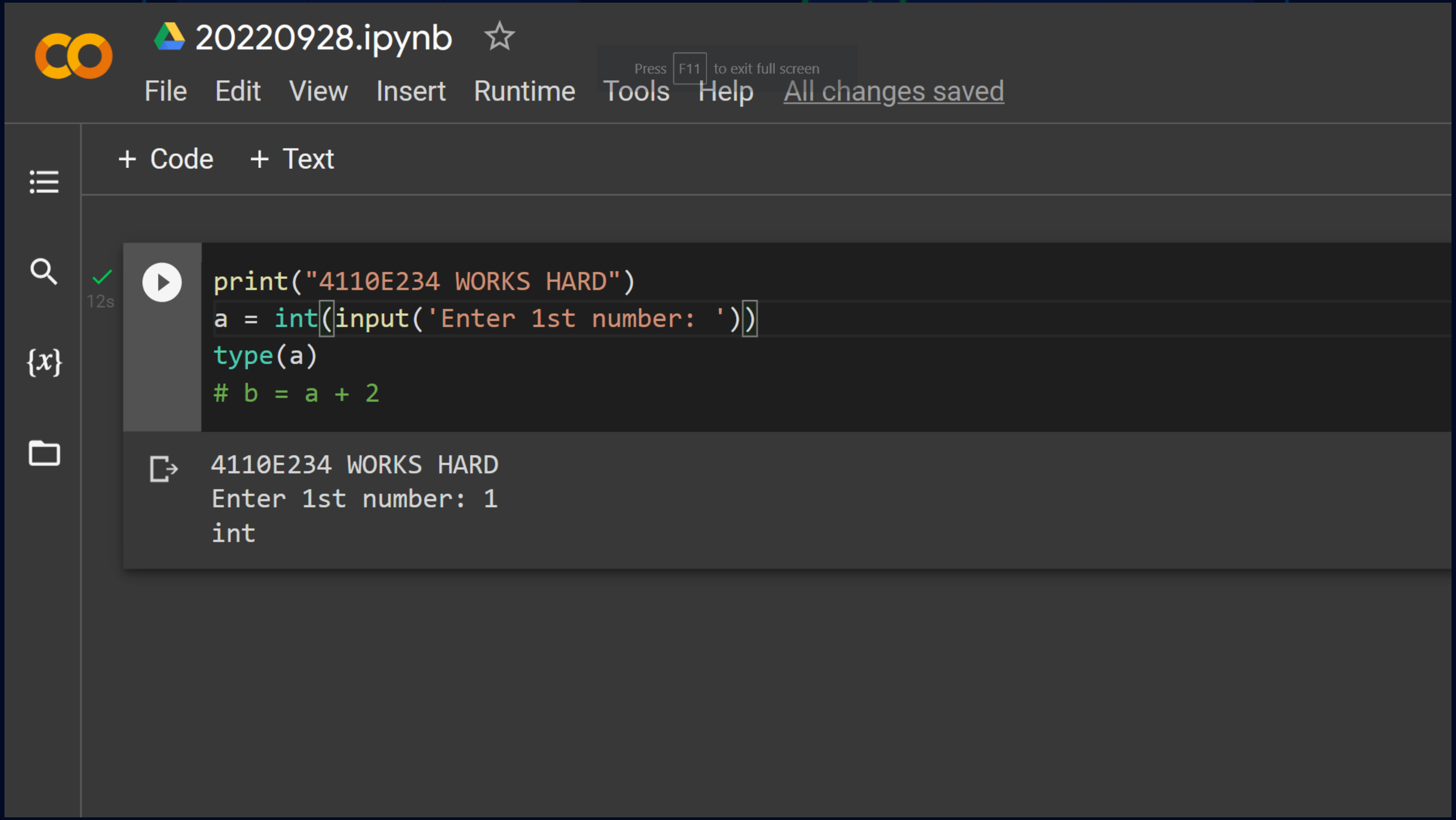


```
print("4110E234 WORKS HARD")
a = input('Enter 1st number: ')
b = a + 2
```

```
4110E234 WORKS HARD
Enter 1st number: 1
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-c555964acf6e> in <module>
      1 print("4110E234 WORKS HARD")
      2 a = input('Enter 1st number: ')
----> 3 b = a + 2

TypeError: can only concatenate str (not "int") to str
```

Data type

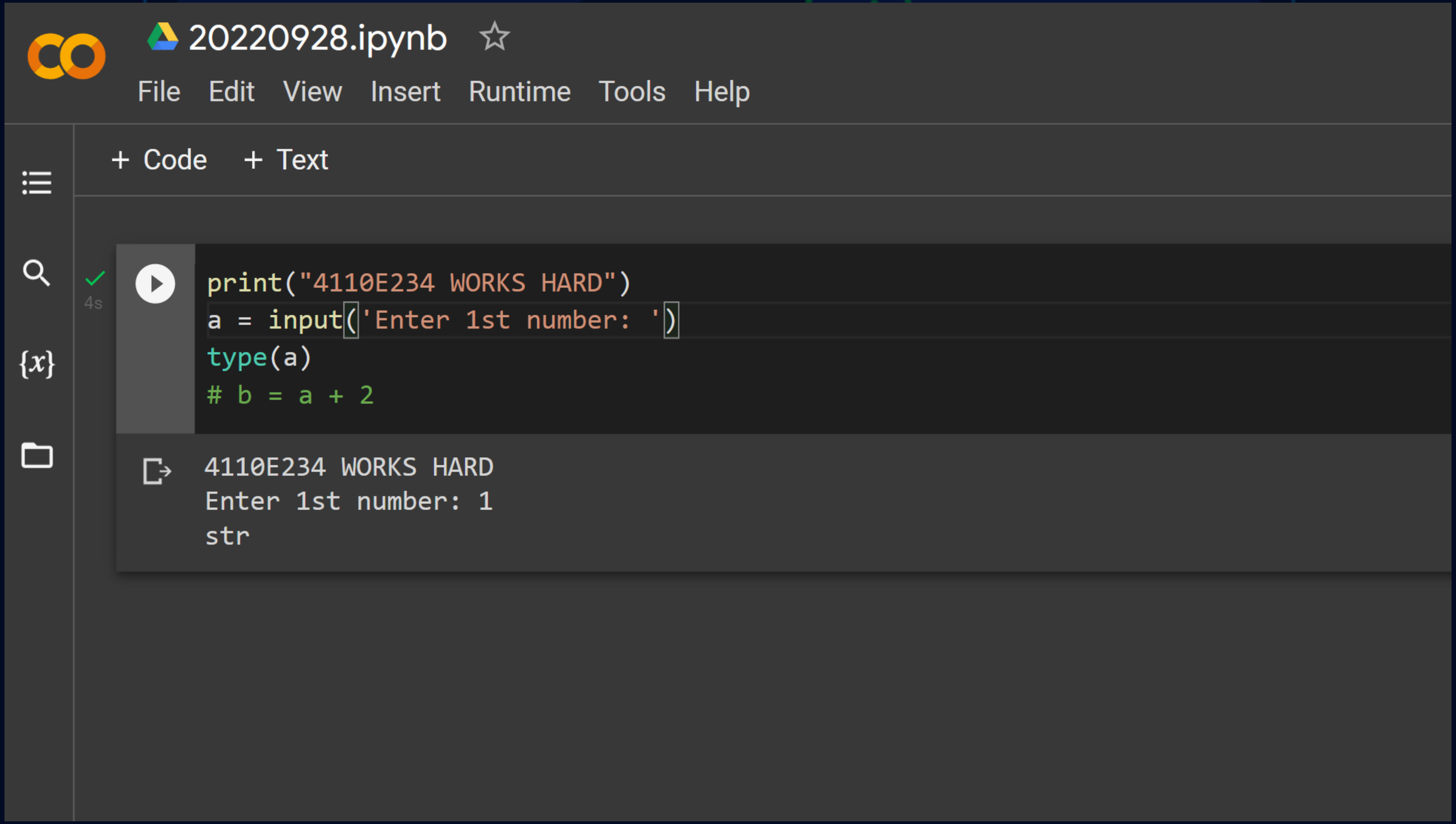
Way: 1


20220928.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Press F11 to exit full screen
+ Code + Text
12s

```
print("4110E234 WORKS HARD")
a = int(input('Enter 1st number: '))
type(a)
# b = a + 2
```

```
4110E234 WORKS HARD
Enter 1st number: 1
int
```

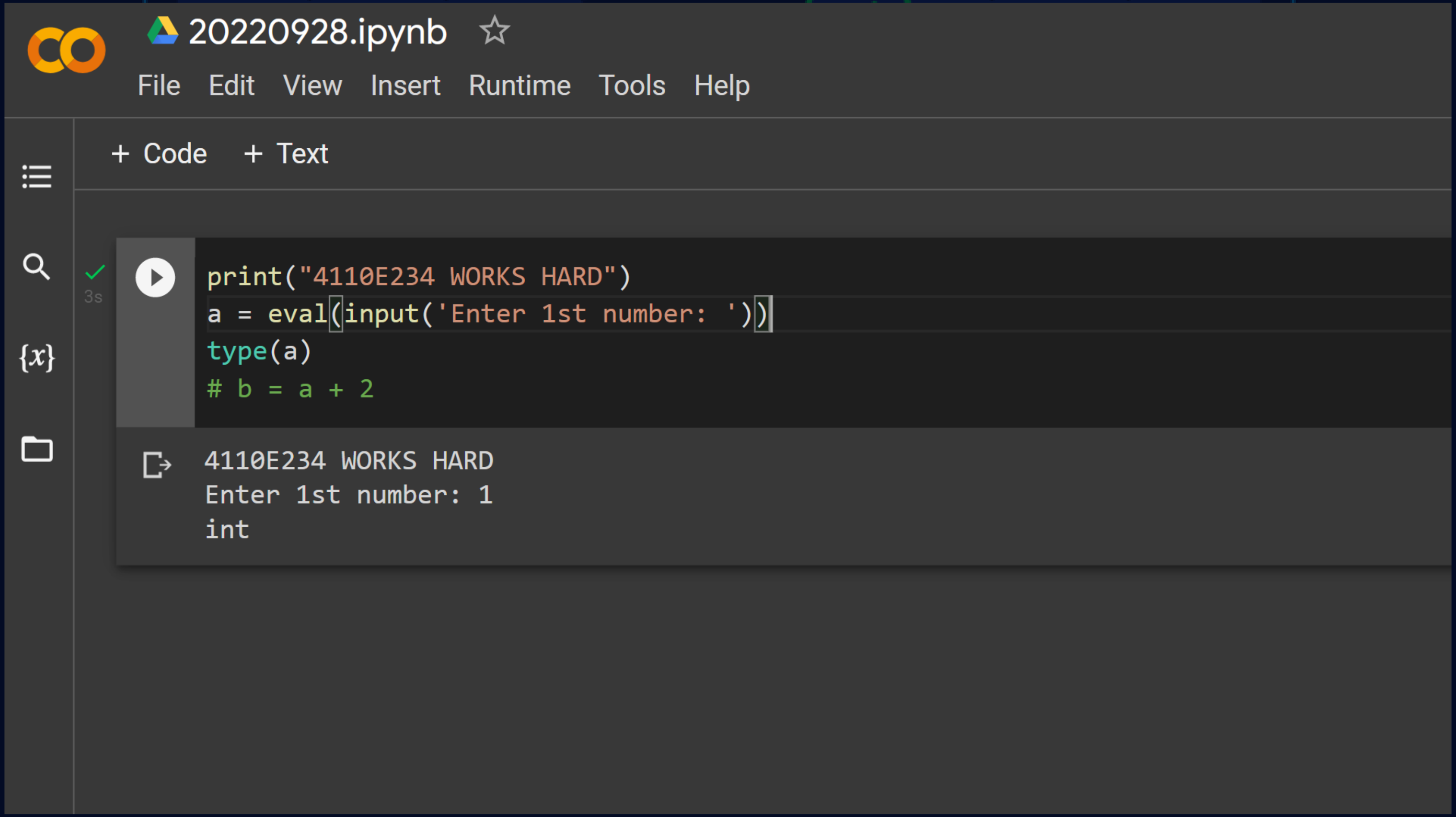
Way: 2



```
print("4110E234 WORKS HARD")
a = input('Enter 1st number: ')
type(a)
# b = a + 2
```

```
4110E234 WORKS HARD
Enter 1st number: 1
str
```

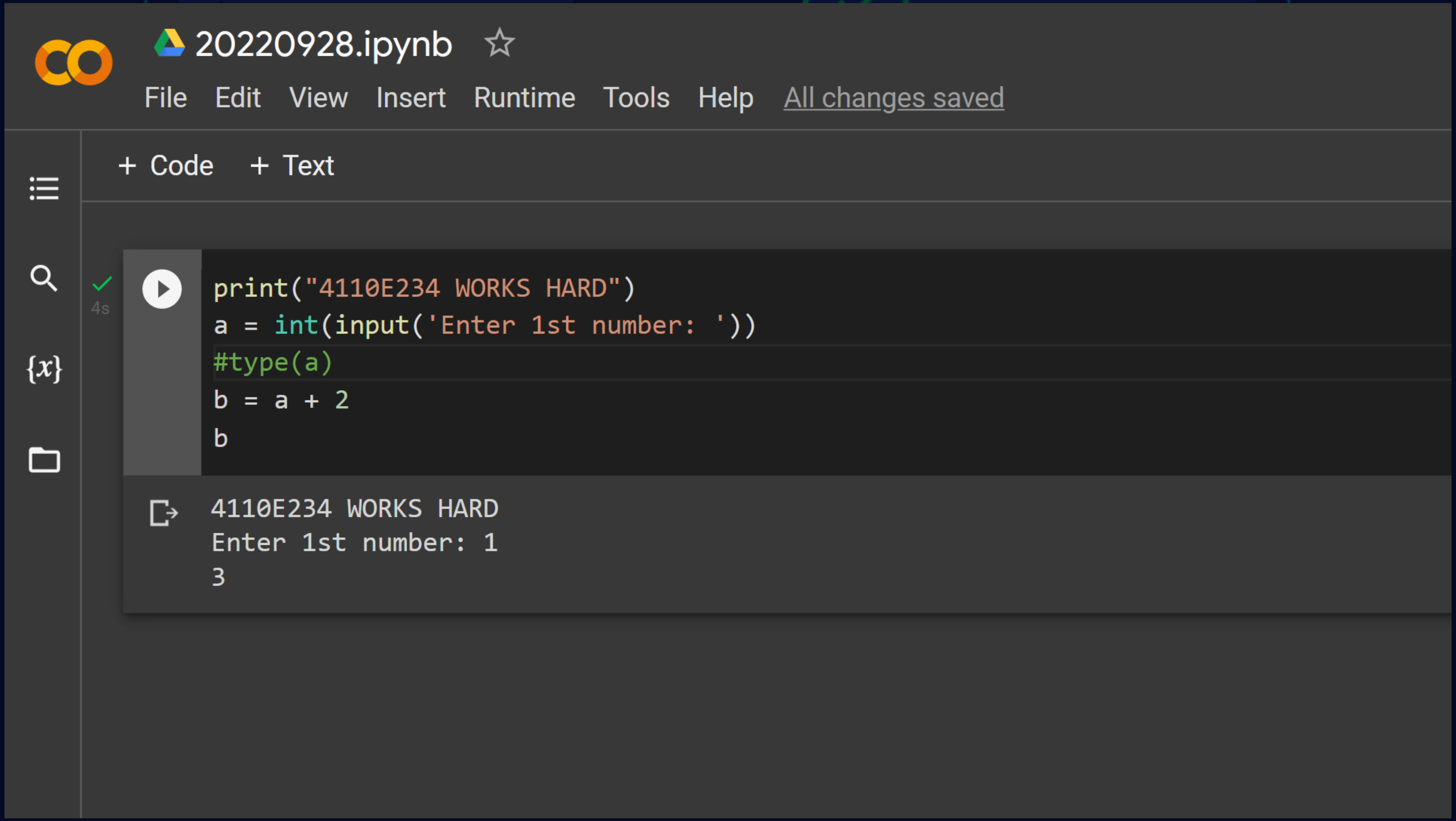
Way: 3



```
print("4110E234 WORKS HARD")
a = eval(input('Enter 1st number: '))
type(a)
# b = a + 2
```

```
4110E234 WORKS HARD
Enter 1st number: 1
int
```

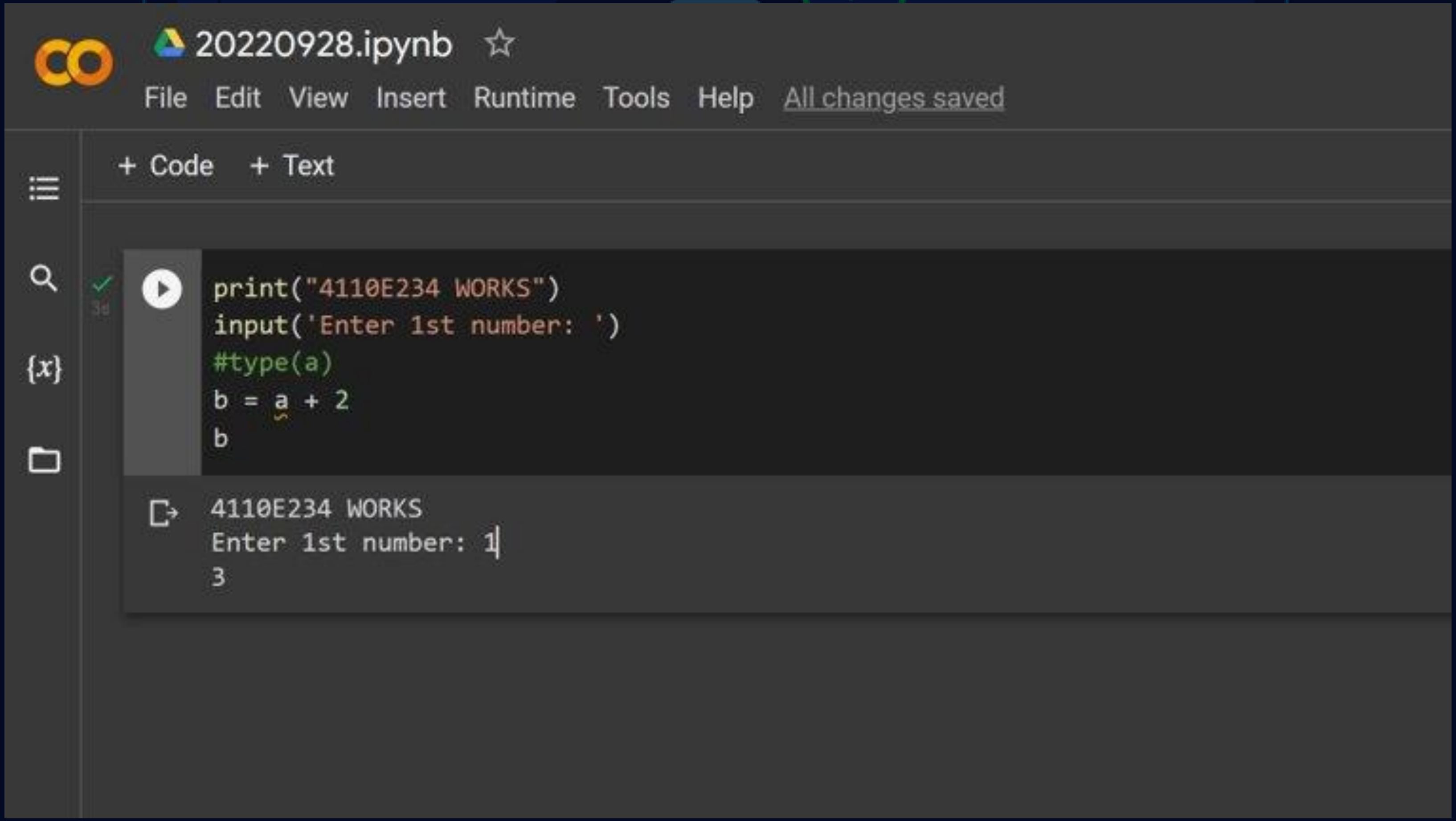
Answer: 1



```python
print("4110E234 WORKS HARD")
a = int(input('Enter 1st number: '))
#type(a)
b = a + 2
b
```

```
4110E234 WORKS HARD
Enter 1st number: 1
3
```

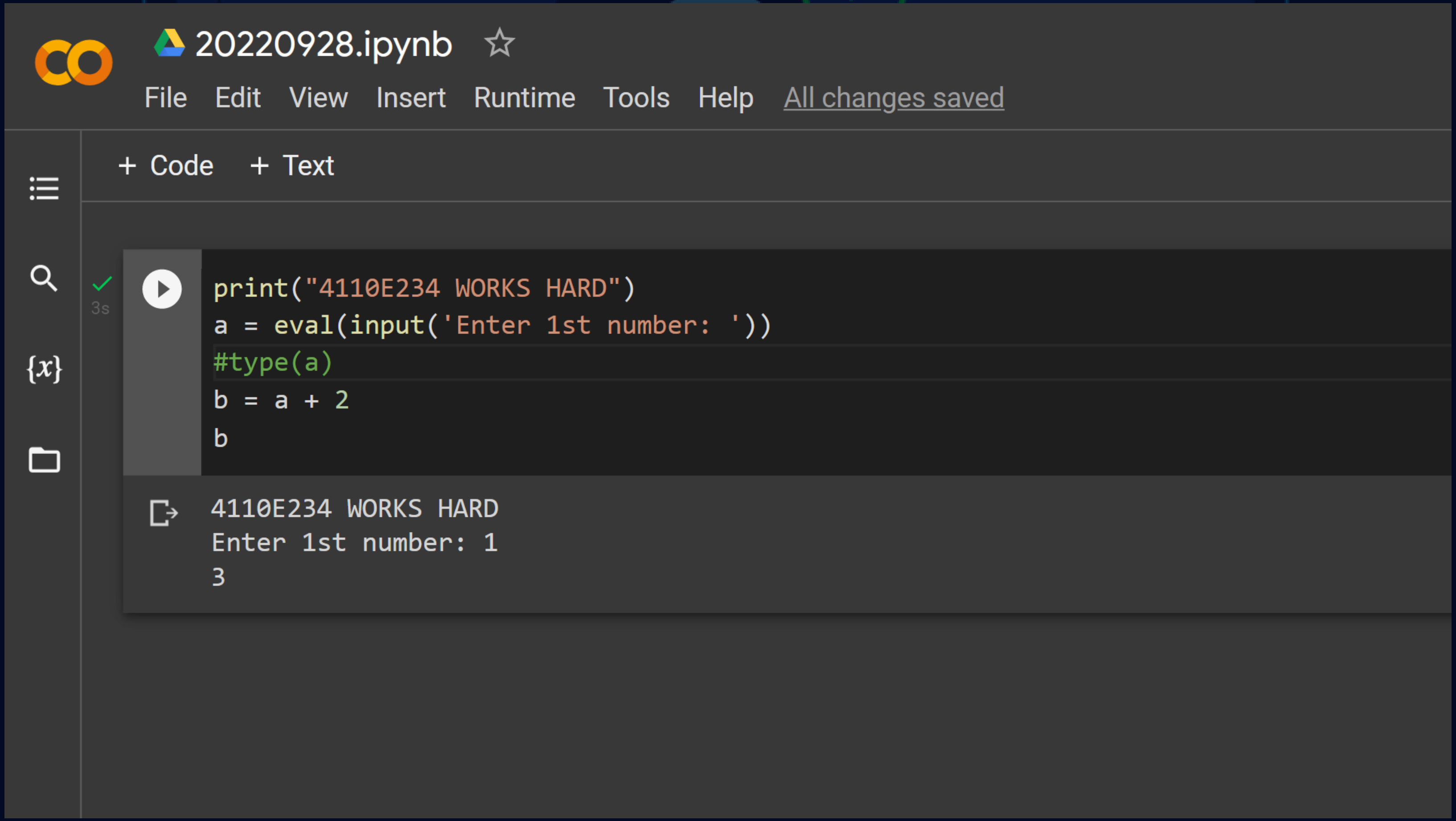
Answer: 2



```
print("4110E234 WORKS")
input('Enter 1st number: ')
#type(a)
b = a + 2
b
```

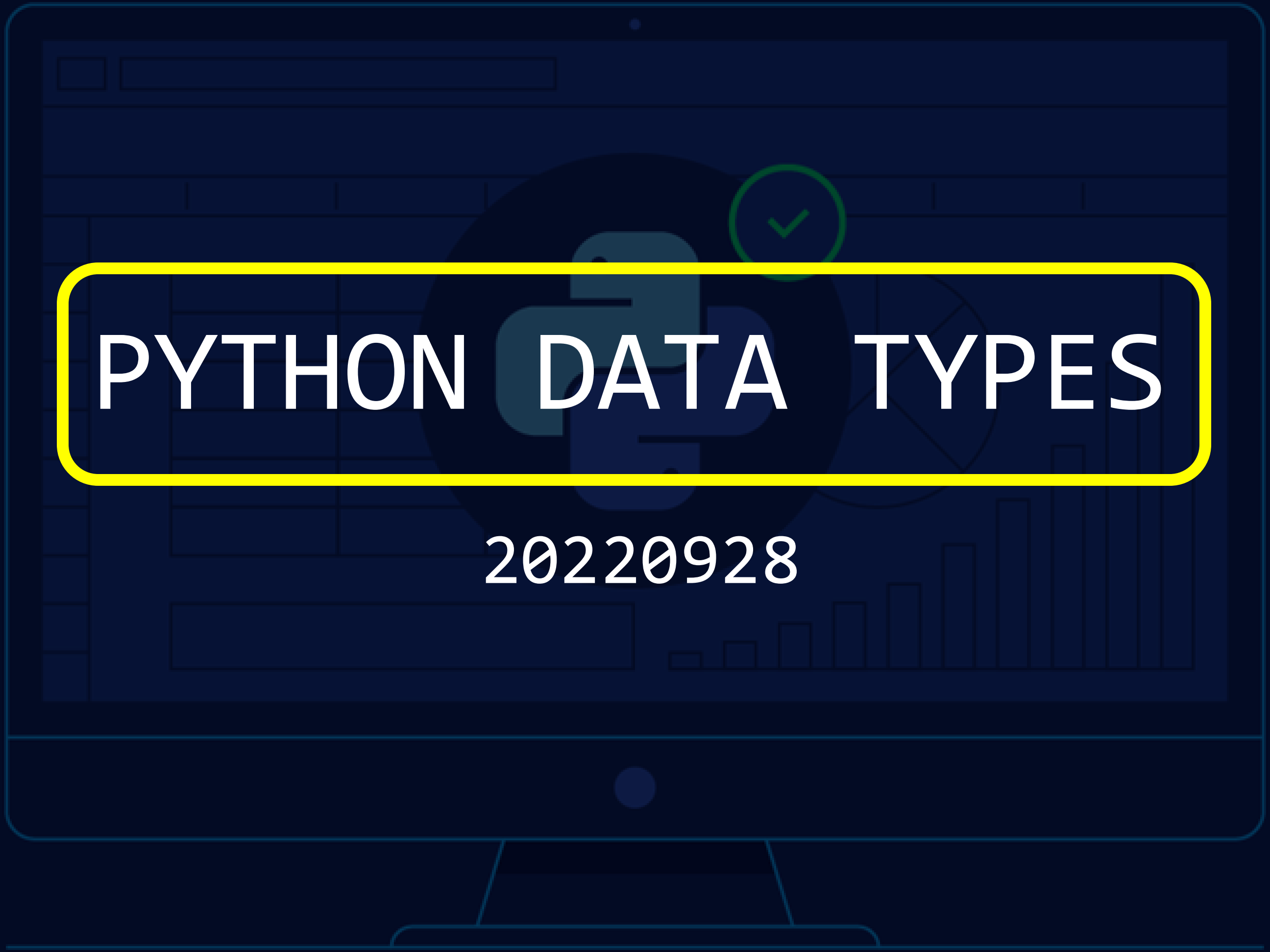
```
4110E234 WORKS
Enter 1st number: 1
3
```

Answer: 3



```
print("4110E234 WORKS HARD")
a = eval(input('Enter 1st number: '))
#type(a)
b = a + 2
b
```

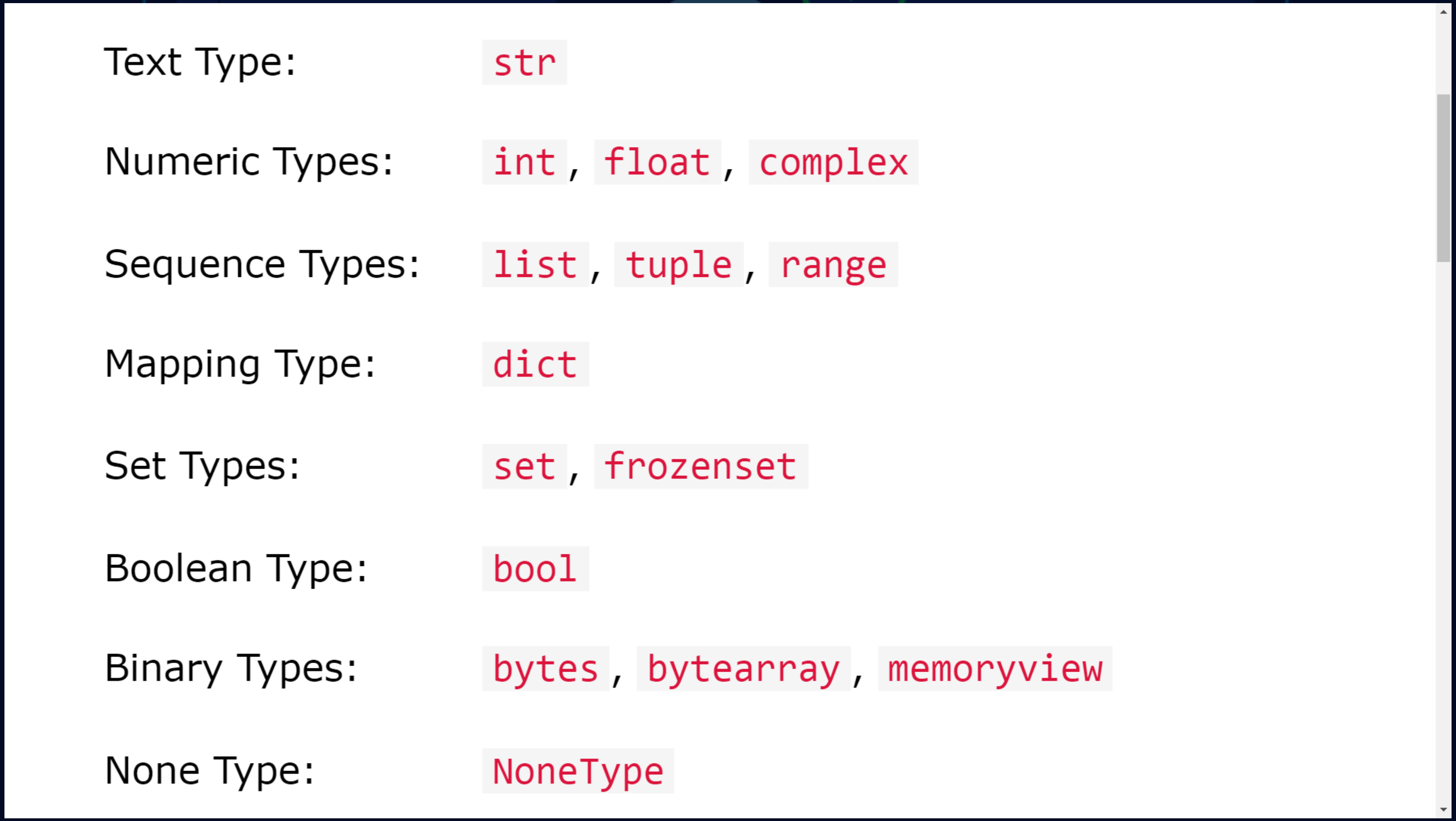
```
4110E234 WORKS HARD
Enter 1st number: 1
3
```



PYTHON DATA TYPES

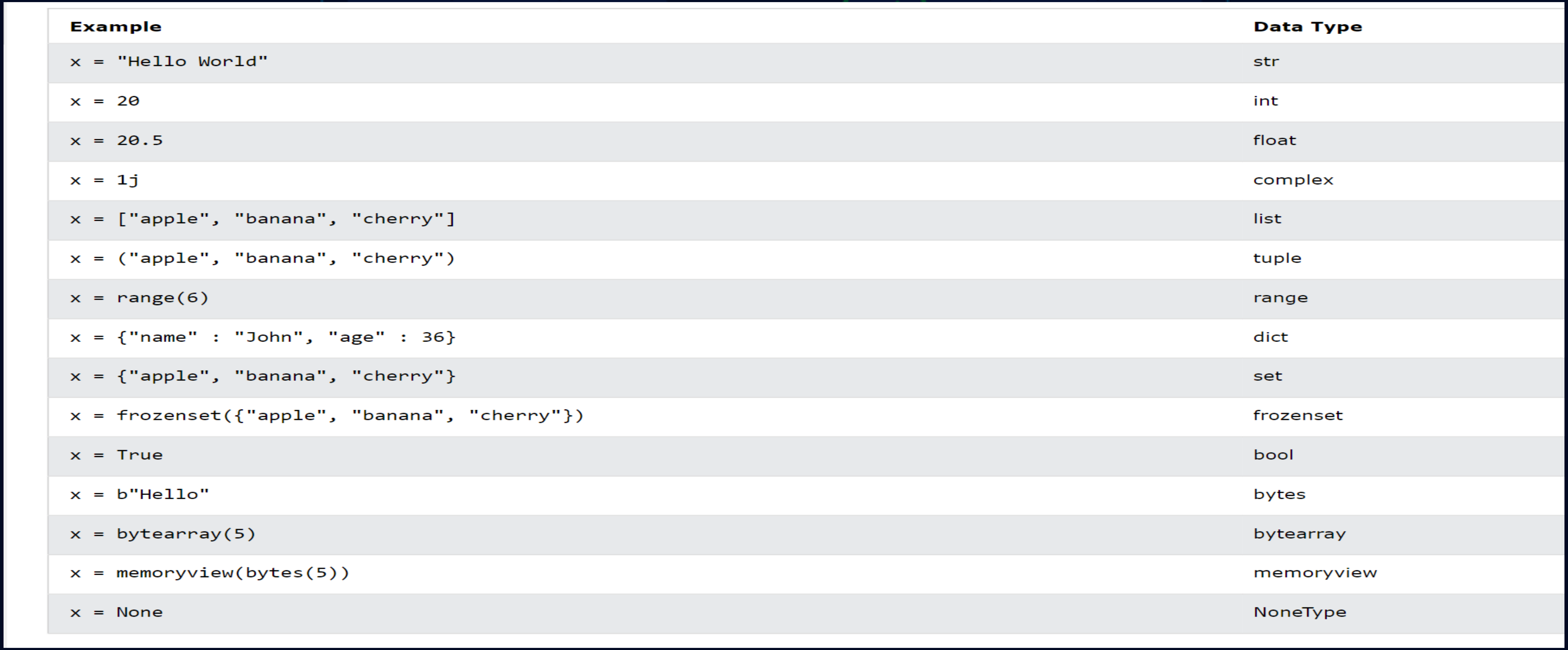
20220928

Data Types



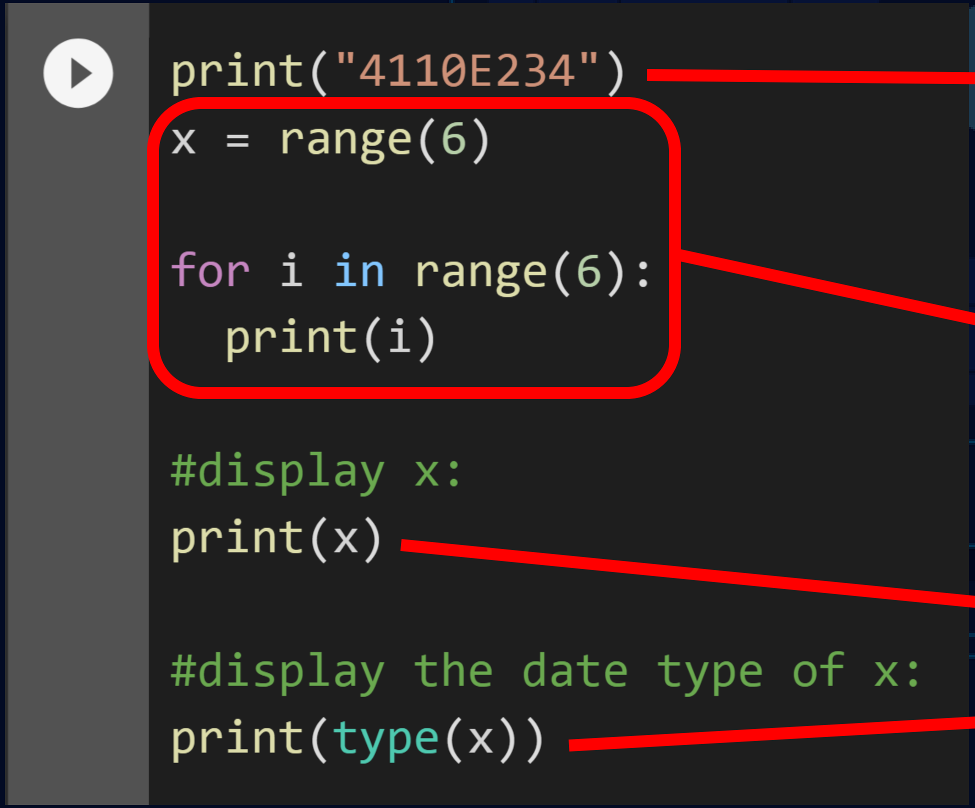
Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

Setting the Specific Data Type

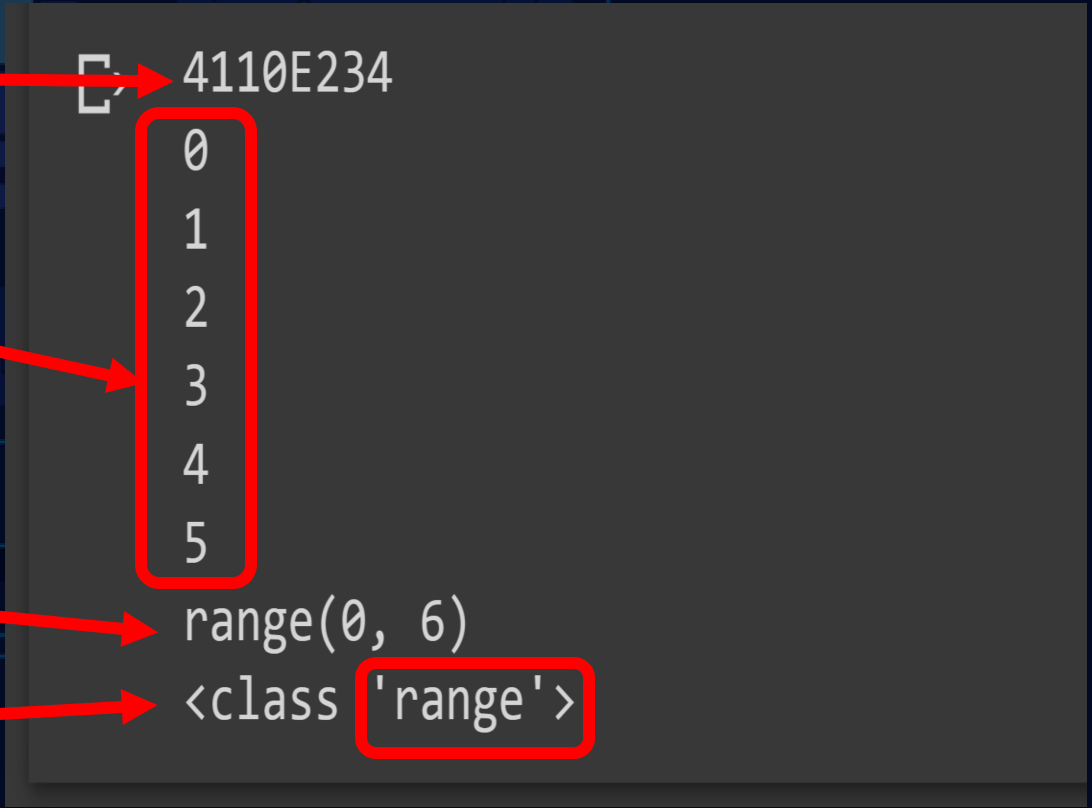


Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview
`x = None`	NoneType

Python Data

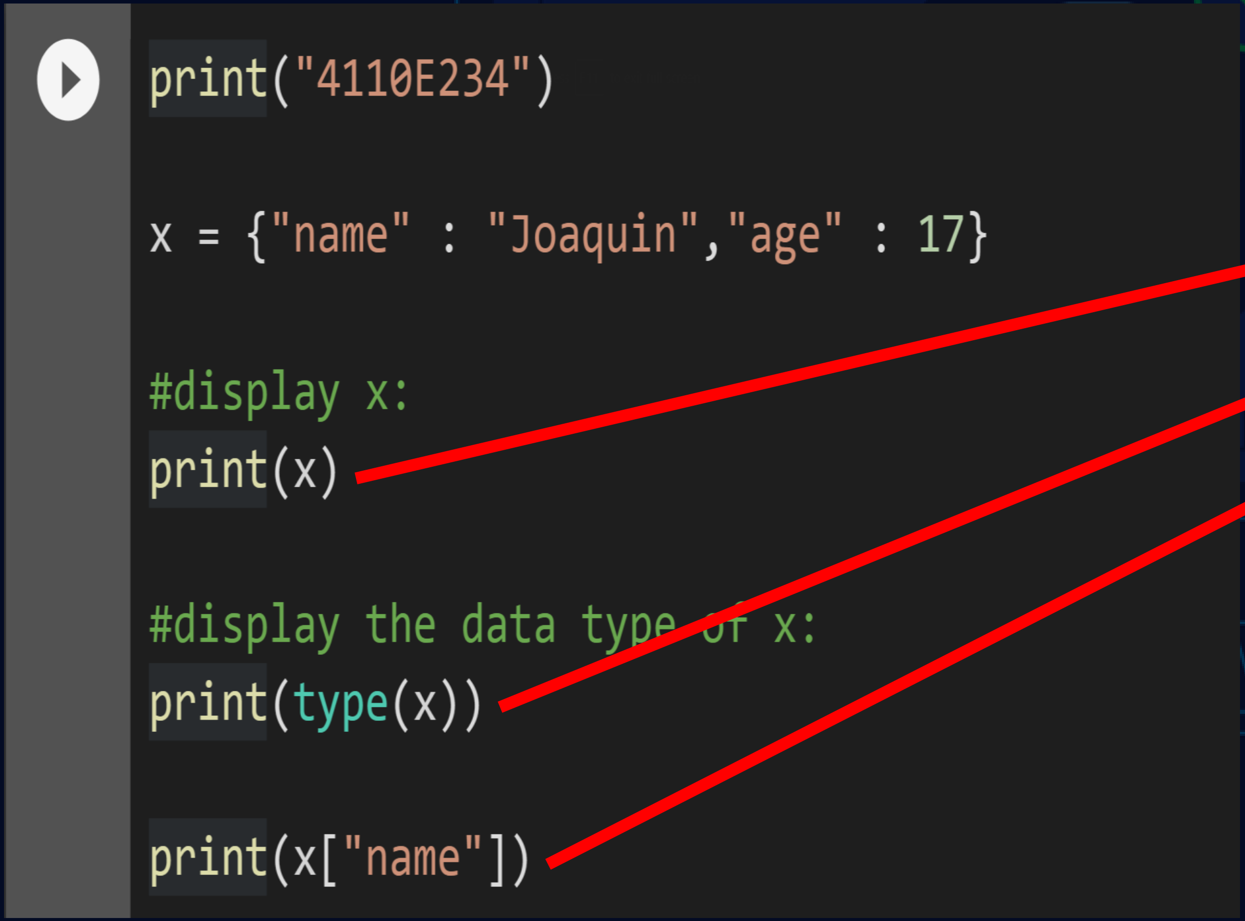


```
print("4110E234")
x = range(6)

for i in range(6):
  print(i)

#display x:
print(x)

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

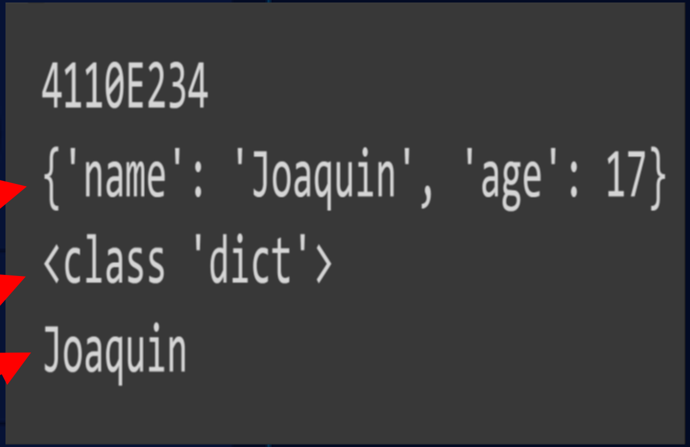


```
4110E234
0
1
2
3
4
5
range(0, 6)
<class 'range'>
```

dict: key-value pair



```
print("4110E234")

x = {"name" : "Joaquin","age" : 17}

#display x:
print(x)

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

print(x["name"])
```

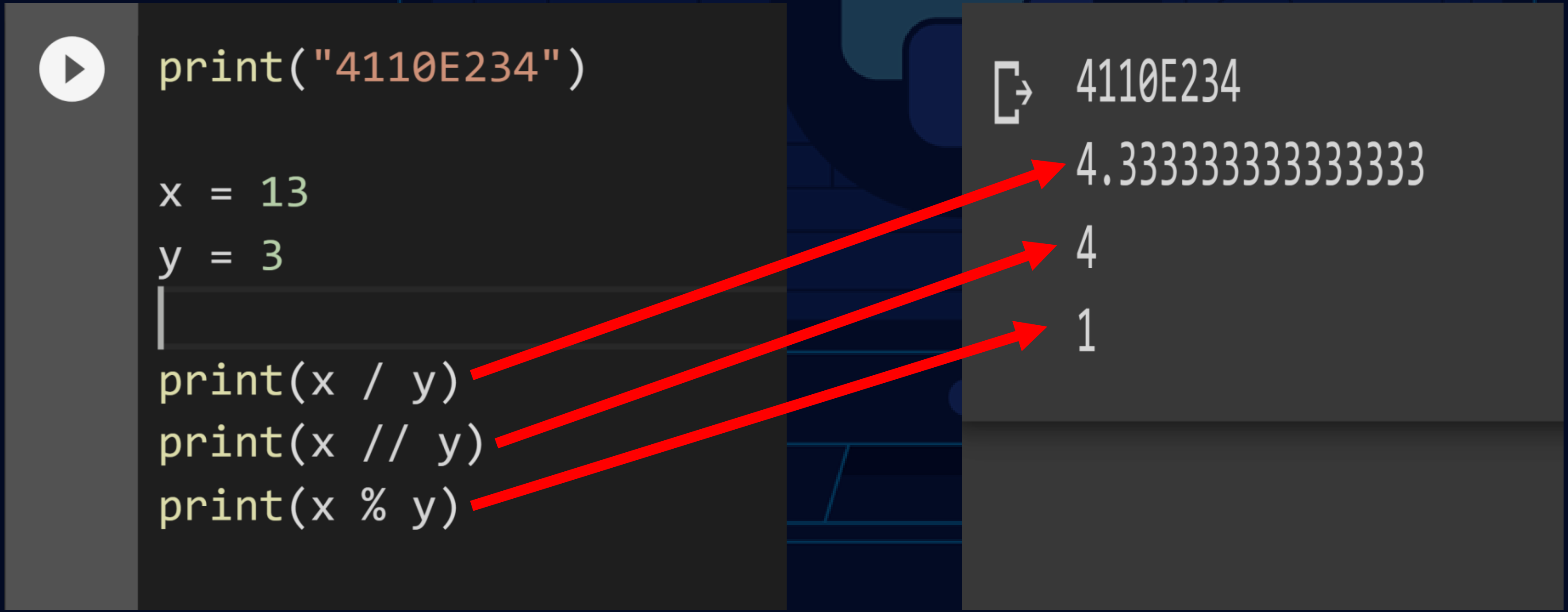


```
4110E234
{'name': 'Joaquin', 'age': 17}
<class 'dict'>
Joaquin
```

PYTHON OPERATORS

20220928

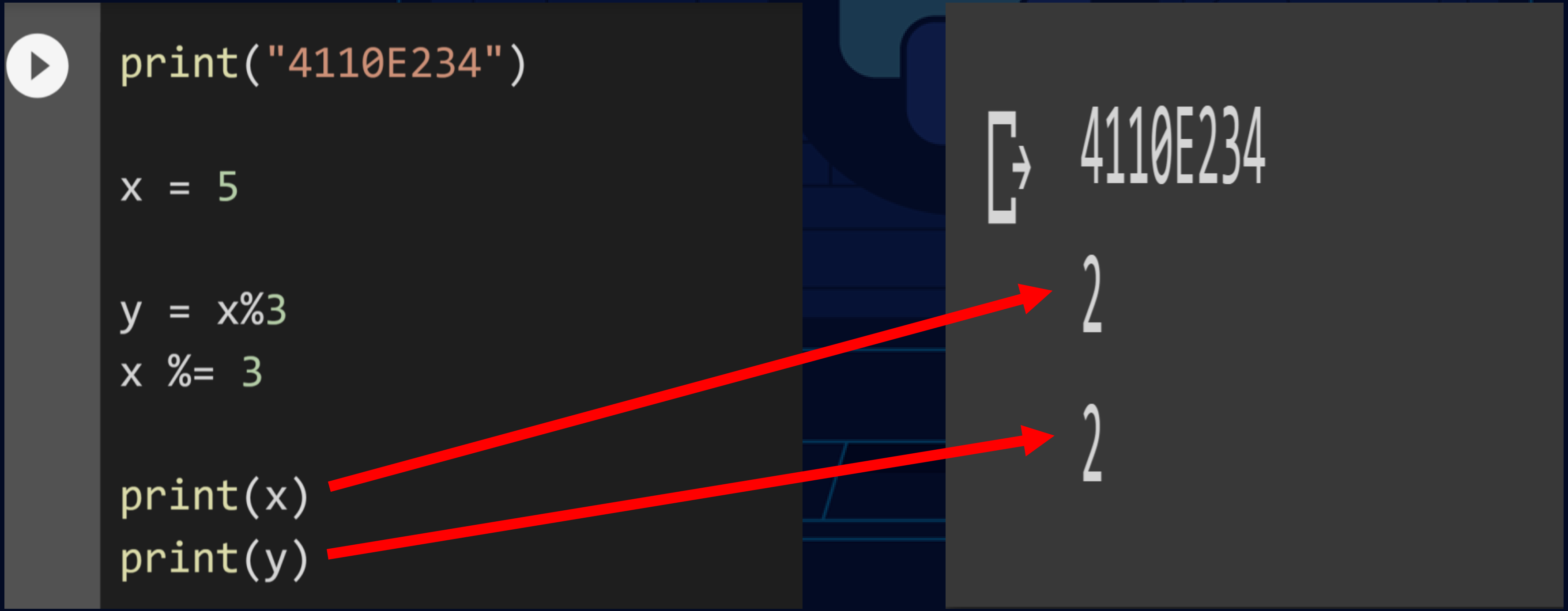
Python Arithmetic Operators



```
print("4110E234")

x = 13
y = 3

print(x / y)
print(x // y)
print(x % y)
```

```
4110E234
4.333333333333333
4
1
```

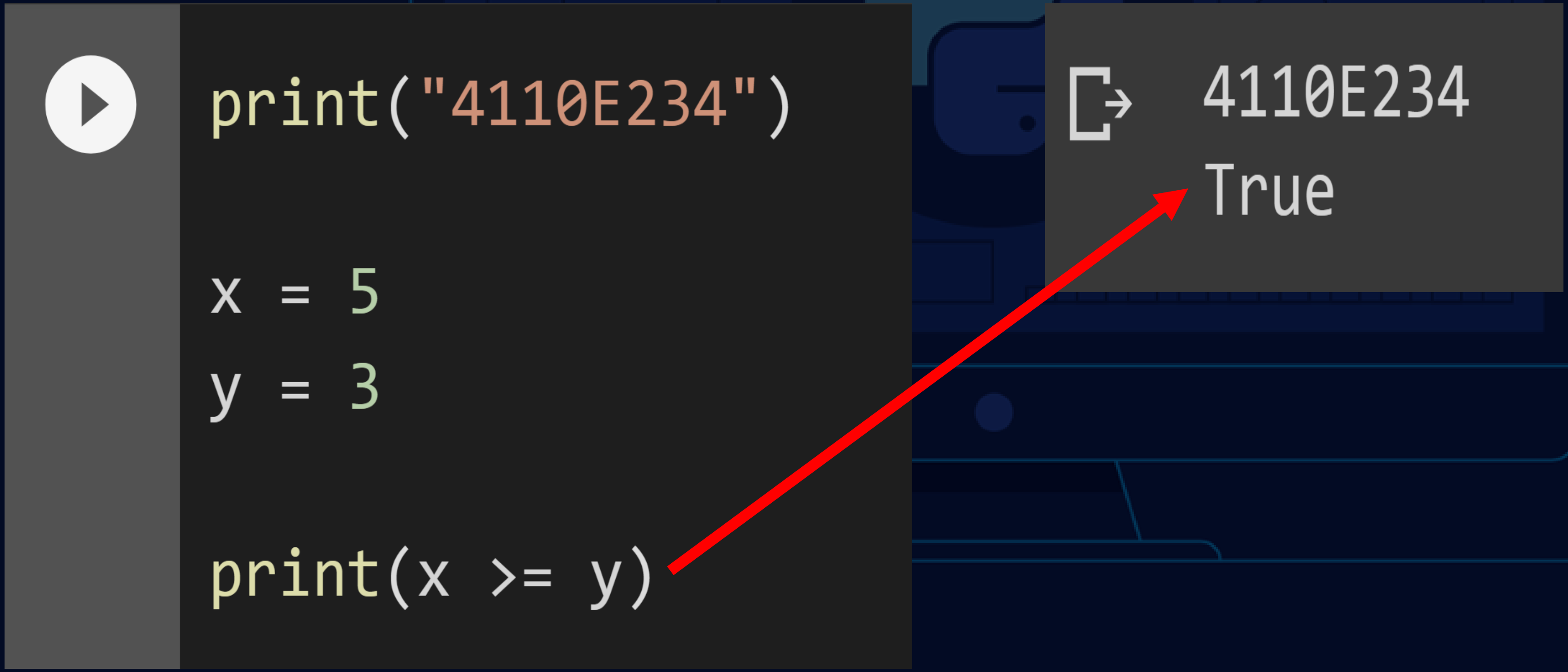
Python Assignment Operators



```
print("4110E234")

x = 5

y = x%3
x %= 3

print(x)
print(y)
```

```
4110E234
2
2
```

Python Comparison Operators

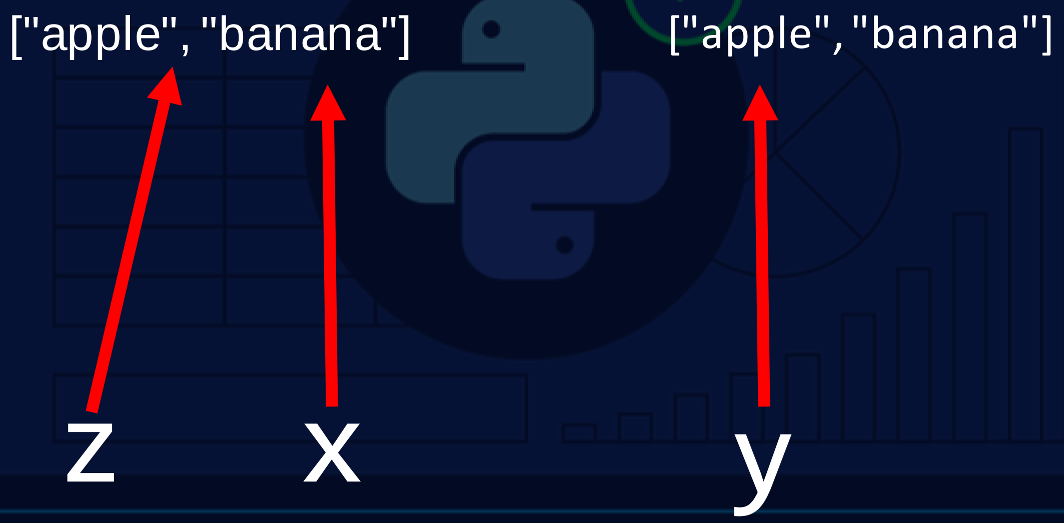


```python
print("4110E234")

x = 5
y = 3

print(x >= y)
```

```
4110E234
True
```

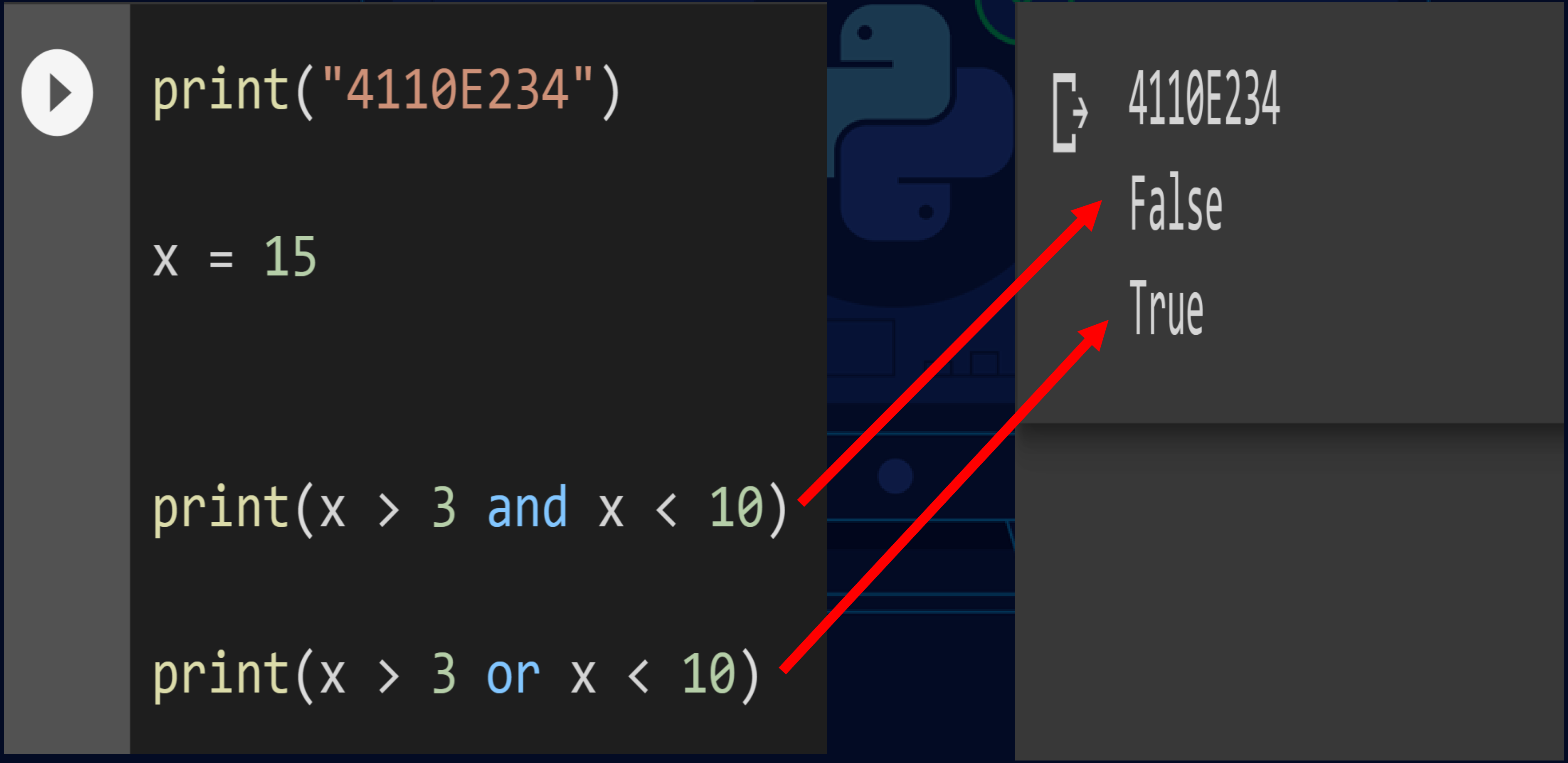
Python Comparison Operators

```
z = x
```



```
x is y == > False
x == y == > True
```

Python Logical Operators

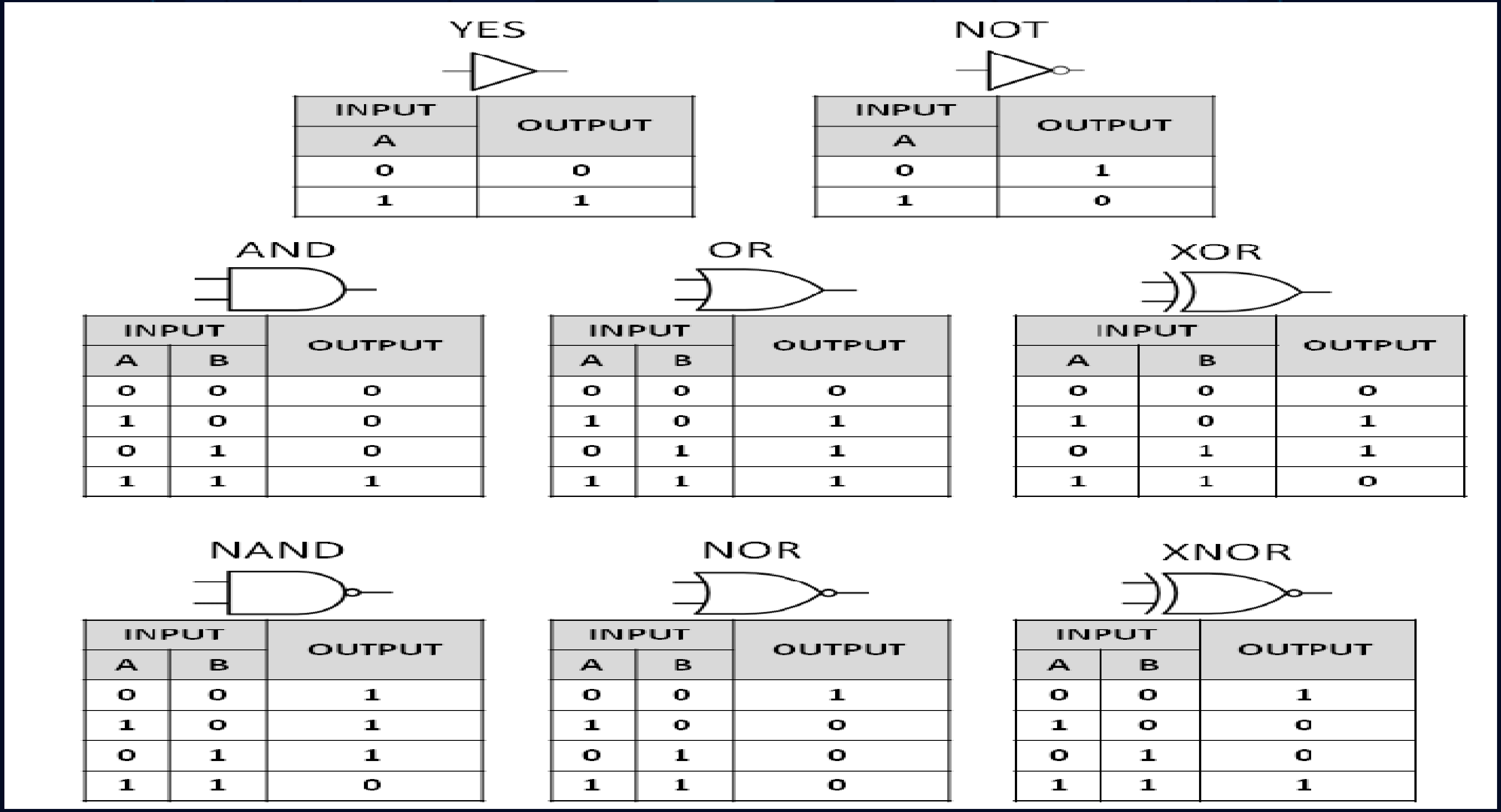


```
print("4110E234")

x = 15


print(x > 3 and x < 10)

print(x > 3 or x < 10)
```

```
4110E234
False
True
```

Python Logic Gate

1 == > TRUE
0 == > FALSE



YES

INPUT	OUTPUT
A	
0	0
1	1

NOT

INPUT	OUTPUT
A	
0	1
1	0

AND

INPUT		OUTPUT
A	B	
0	0	0
1	0	0
0	1	0
1	1	1

OR

INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	1

XOR

INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	0

NAND

INPUT		OUTPUT
A	B	
0	0	1
1	0	1
0	1	1
1	1	0

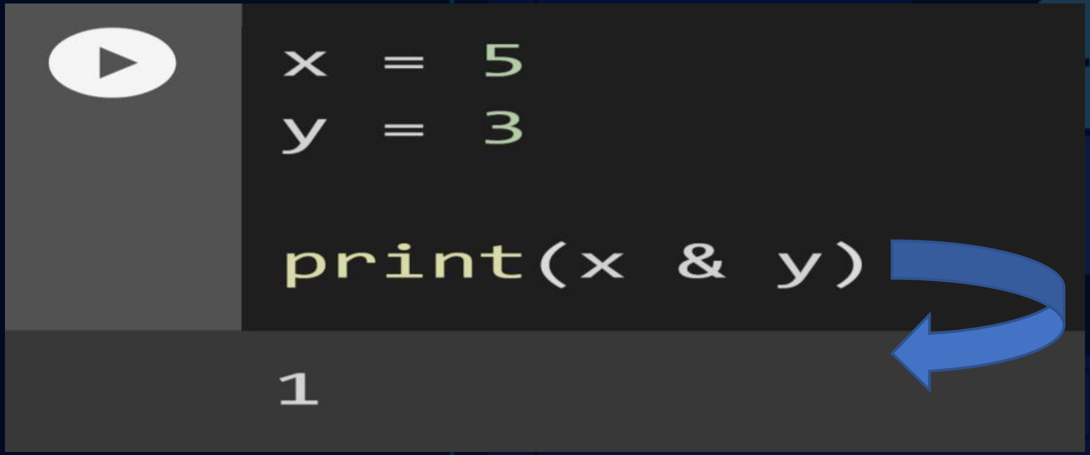
NOR

INPUT		OUTPUT
A	B	
0	0	1
1	0	0
0	1	0
1	1	0

XNOR

INPUT		OUTPUT
A	B	
0	0	1
1	0	0
0	1	0
1	1	1

Python Bitewise Operators



```
x = 5
y = 3

print(x & y)
```

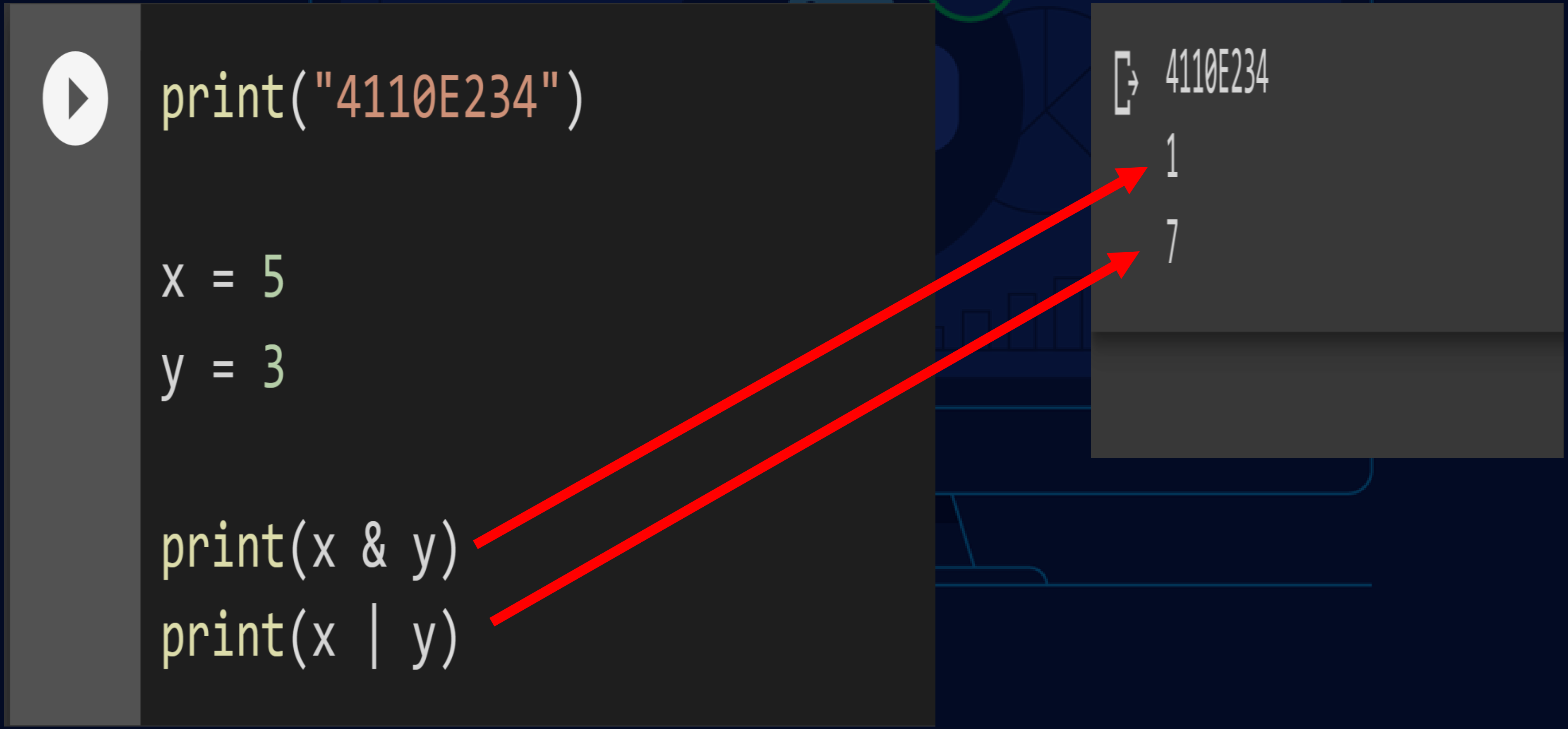
```
1
```

binary

1*22+0*21+1*20

- x = 5 (101)2
- y = 3 (011)2
- print(x & y) (001)2

Python Bitewise Operators



```
print("4110E234")

x = 5
y = 3

print(x & y)
print(x | y)
```

```
4110E234
1
7
```

THANK YOU!

CHINESE NAME: 喬萬斯

STUDENT ID: 4110E234

English Name: Joaquin Vasti R. Rapada

nickname: Wax

email: vastiplayer@gmail.com

GITHUB: https://github.com/4110E34