

# 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.

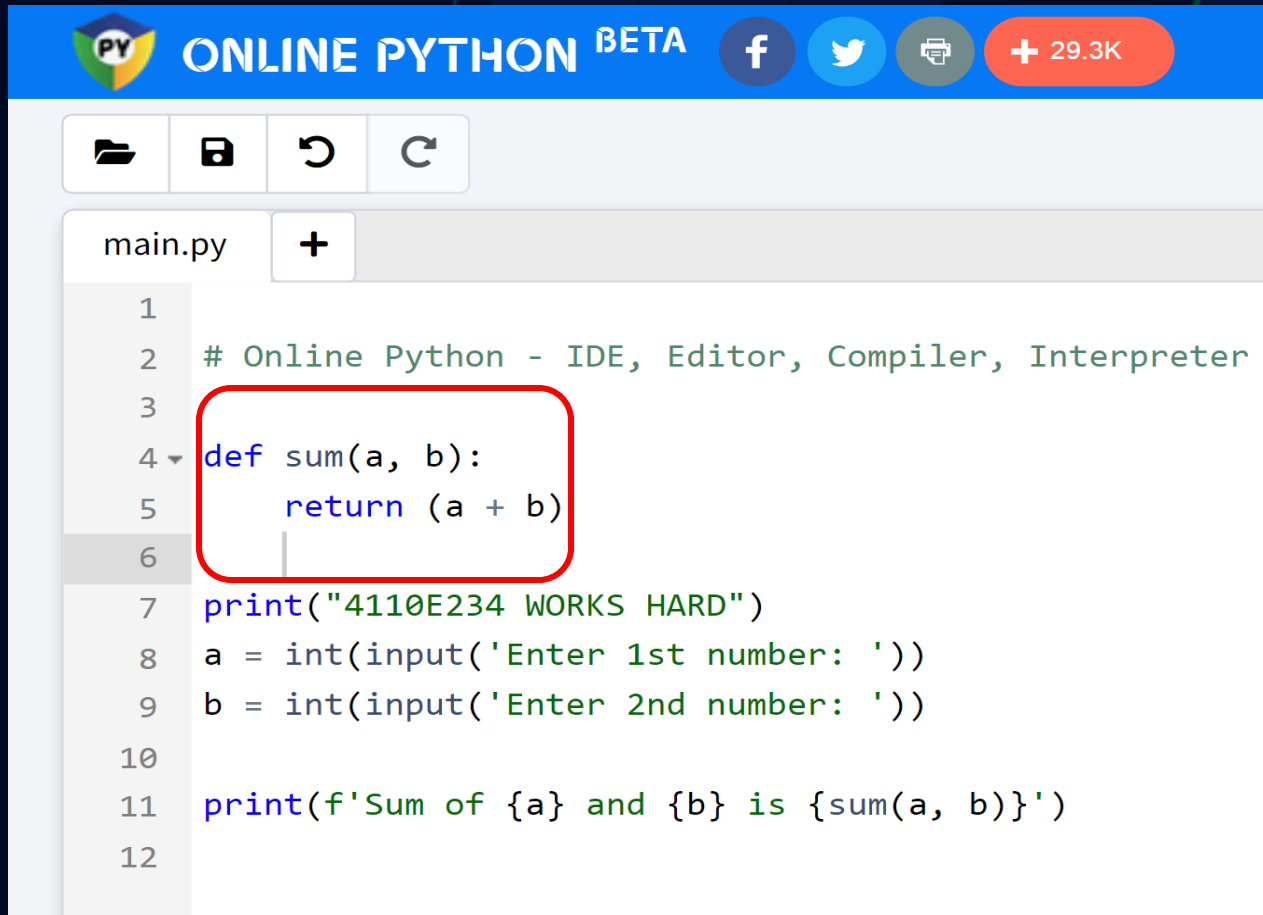
## The Top 10 Reasons

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.

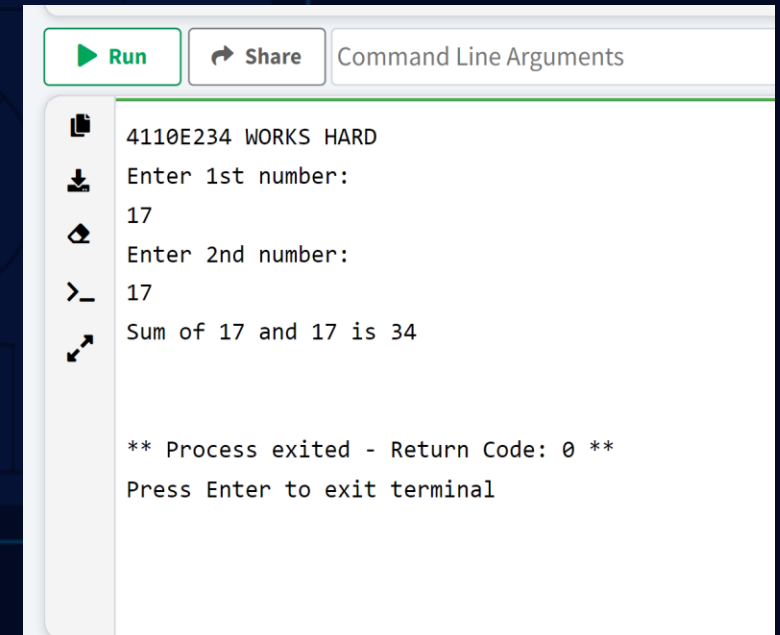
# Python Online Interpreter: 1



The screenshot shows the 'Online Python BETA' web interface. At the top, there's a blue header with the 'PY' logo, social media icons for Facebook, Twitter, and Print, and a red button with '+ 29.3K'. Below the header is a toolbar with icons for file operations. The main area is a code editor with a file named 'main.py'. The code is as follows:

```
1
2 # Online Python - IDE, Editor, Compiler, Interpreter
3
4 def sum(a, b):
5     return (a + b)
6
7 print("4110E234 WORKS HARD")
8 a = int(input('Enter 1st number: '))
9 b = int(input('Enter 2nd number: '))
10
11 print(f'Sum of {a} and {b} is {sum(a, b)}')
12
```

The function definition on lines 4 and 5 is highlighted with a red rectangle.




The screenshot shows the terminal output of the Python script. It includes a 'Run' button, a 'Share' button, and a 'Command Line Arguments' input field. The output is as follows:


```
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

  
Python Online Compiler



  
main.py

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



Shell

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
```

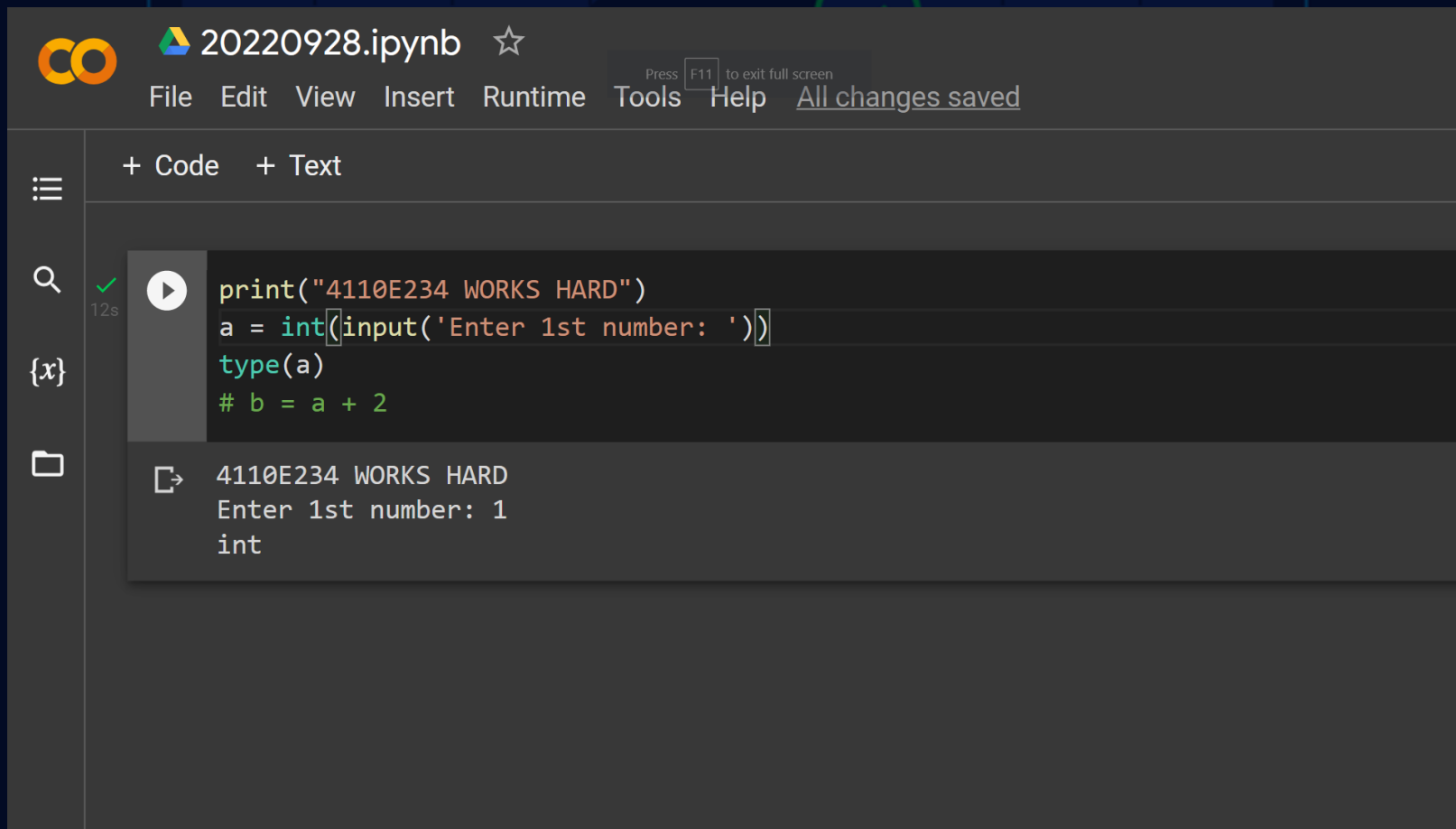
```
4110E234 WORKS HARD  
Enter 1st number: 1
```

```
-----  
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
```

Data type **TypeError:** can only concatenate str (not "int") to str

SEARCH STACK OVERFLOW

# Way: 1



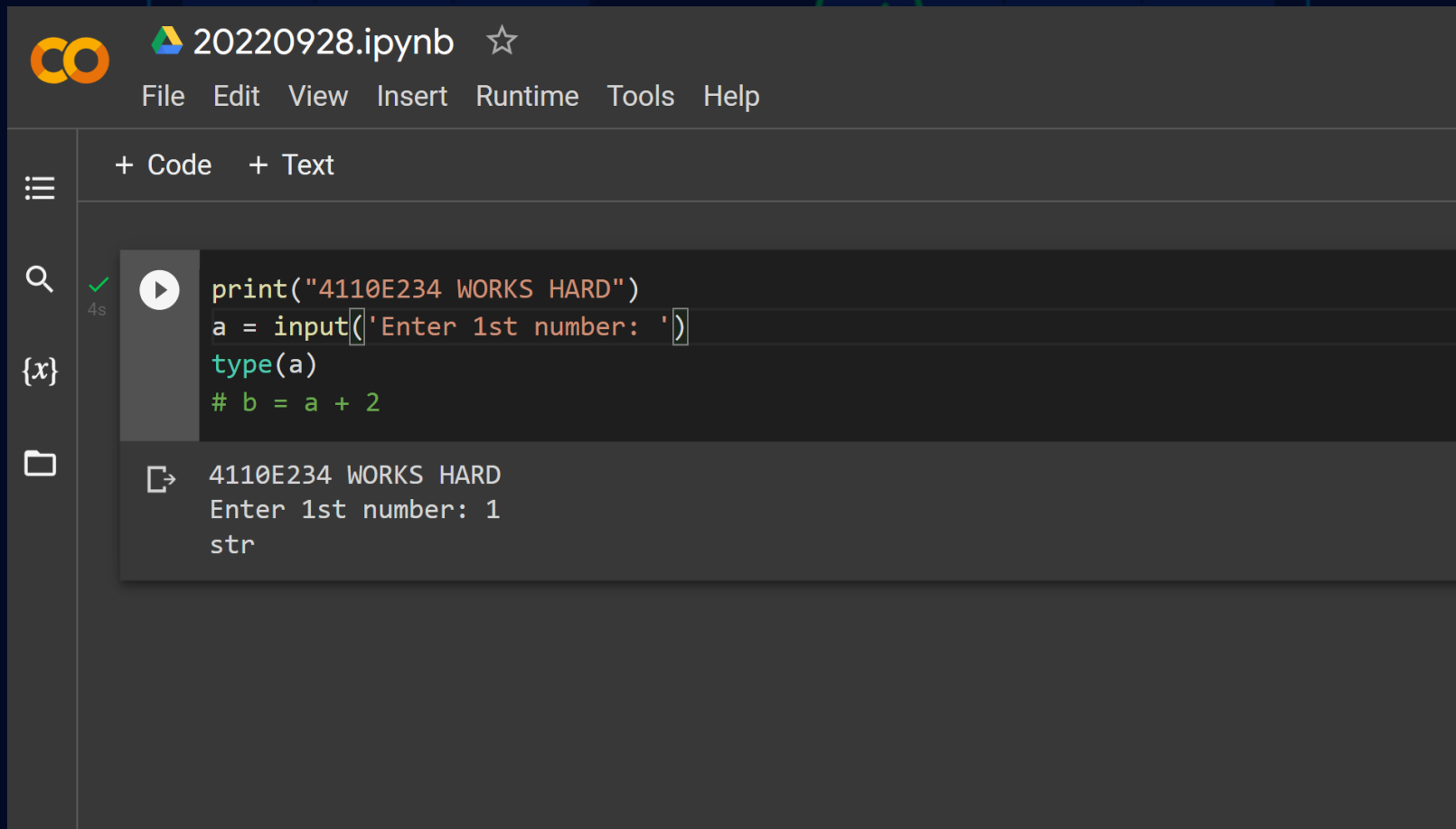
The screenshot shows a Jupyter Notebook window titled "20220928.ipynb". The interface includes a top menu bar with options: File, Edit, View, Insert, Runtime, Tools, Help, and a status bar indicating "All changes saved". Below the menu bar, there are buttons for "+ Code" and "+ Text". The main area displays a code cell with the following Python code:

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

To the left of the code cell, there is a play button icon and a green checkmark with "12s" below it. Below the code cell, the output is displayed:

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

# Way: 2



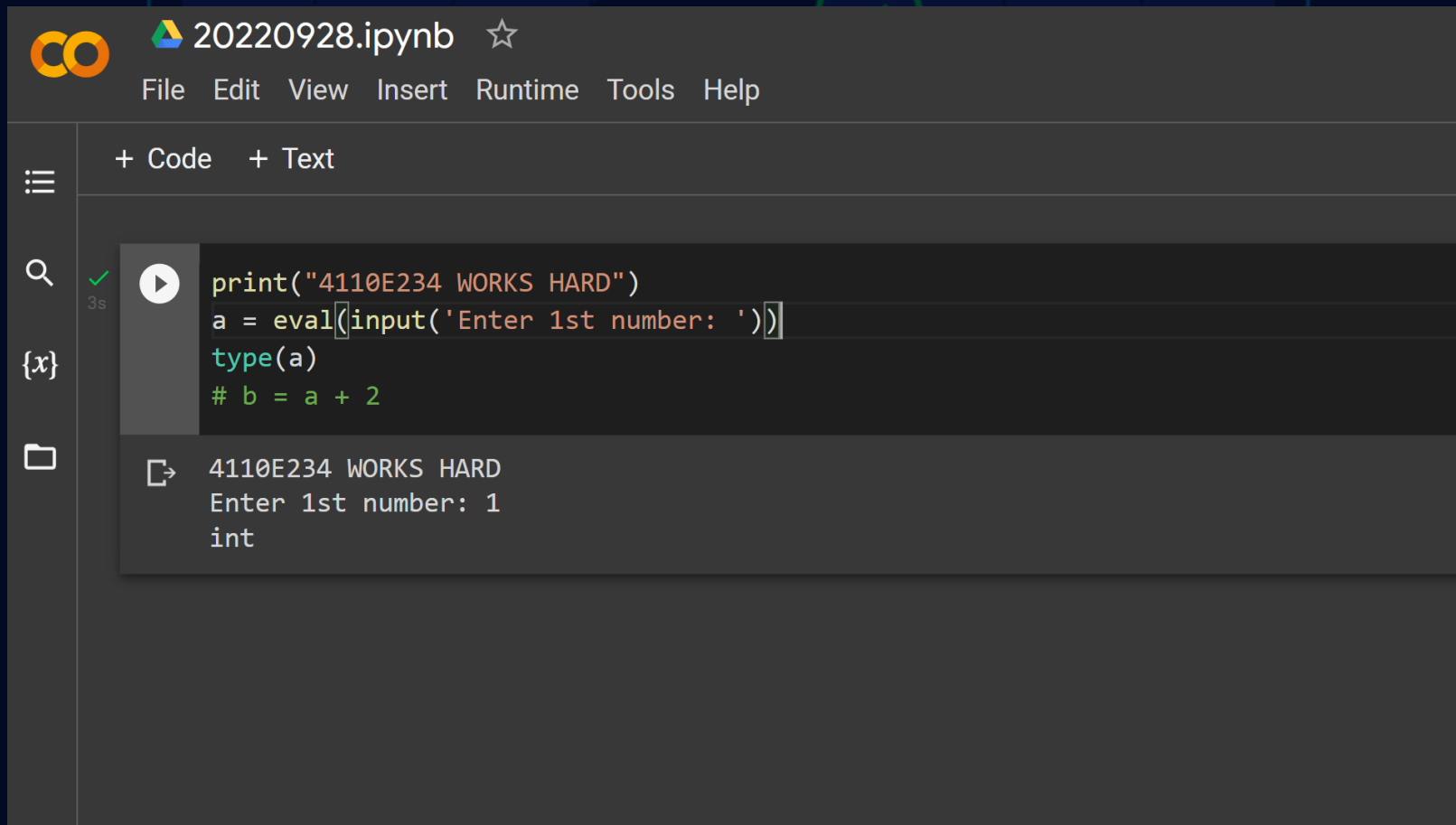
The image shows a JupyterLab interface. At the top, there's a header bar with the CO logo, a file icon, and the filename "20220928.ipynb" with a star icon. Below the header is a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". On the left side, there's a sidebar with icons for a menu, search, variables (showing {x}), and a file explorer. The main area contains a code cell with a play button icon, a green checkmark, and a "4s" timer. The code in the cell is:

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

Below the code cell, the output is displayed:

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

# Way: 3



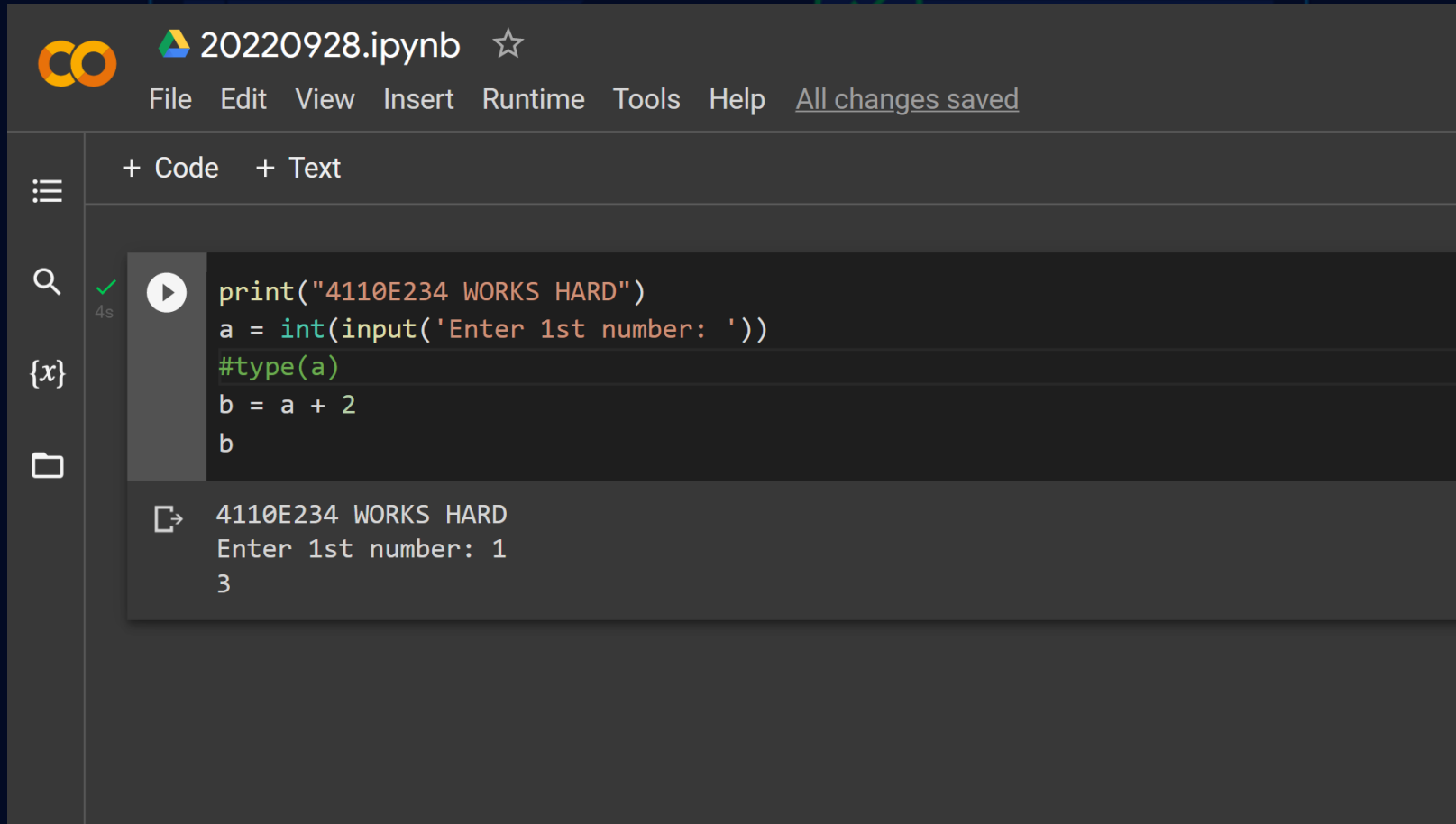
The image shows a Jupyter Notebook interface. At the top, there's a header bar with the Colab logo, the filename "20220928.ipynb", and a star icon. Below this is a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". The left sidebar contains icons for a menu, search, variables, and files. The main area shows a code cell with a play button icon, a green checkmark, and a "3s" execution time. The code in the cell is:

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

Below the code cell, the output is displayed:

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

# Answer: 1

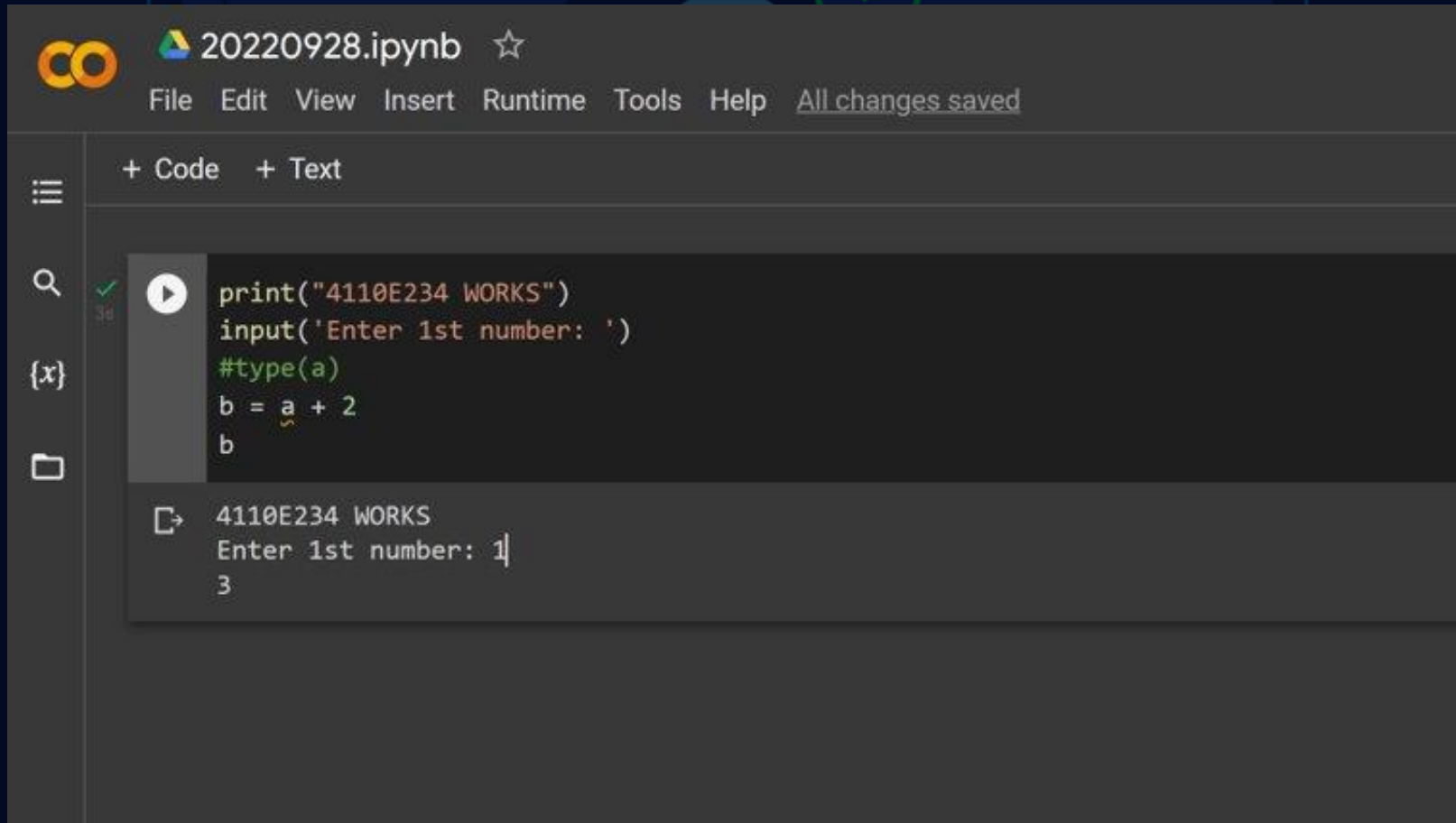


The image shows a Jupyter Notebook interface. At the top, the notebook is titled "20220928.ipynb" with a star icon. Below the title is a menu bar with options: File, Edit, View, Insert, Runtime, Tools, Help, and a link "All changes saved". On the left side, there is a sidebar with icons for a menu, search, variables, and files. The main area contains a code cell with the following Python code:

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

Below the code cell, the output is displayed. It shows the string "4110E234 WORKS HARD", followed by the prompt "Enter 1st number: 1", and the input "3".

# Answer: 2



The image shows a Jupyter Notebook interface. At the top, the title bar reads "20220928.ipynb" with a star icon. Below the title bar is a menu bar with options: File, Edit, View, Insert, Runtime, Tools, Help, and a status message "All changes saved". On the left side, there is a sidebar with icons for a menu, search, variables, and a file explorer. The main area contains a code cell with a play button icon and a green checkmark. The code in the cell is:

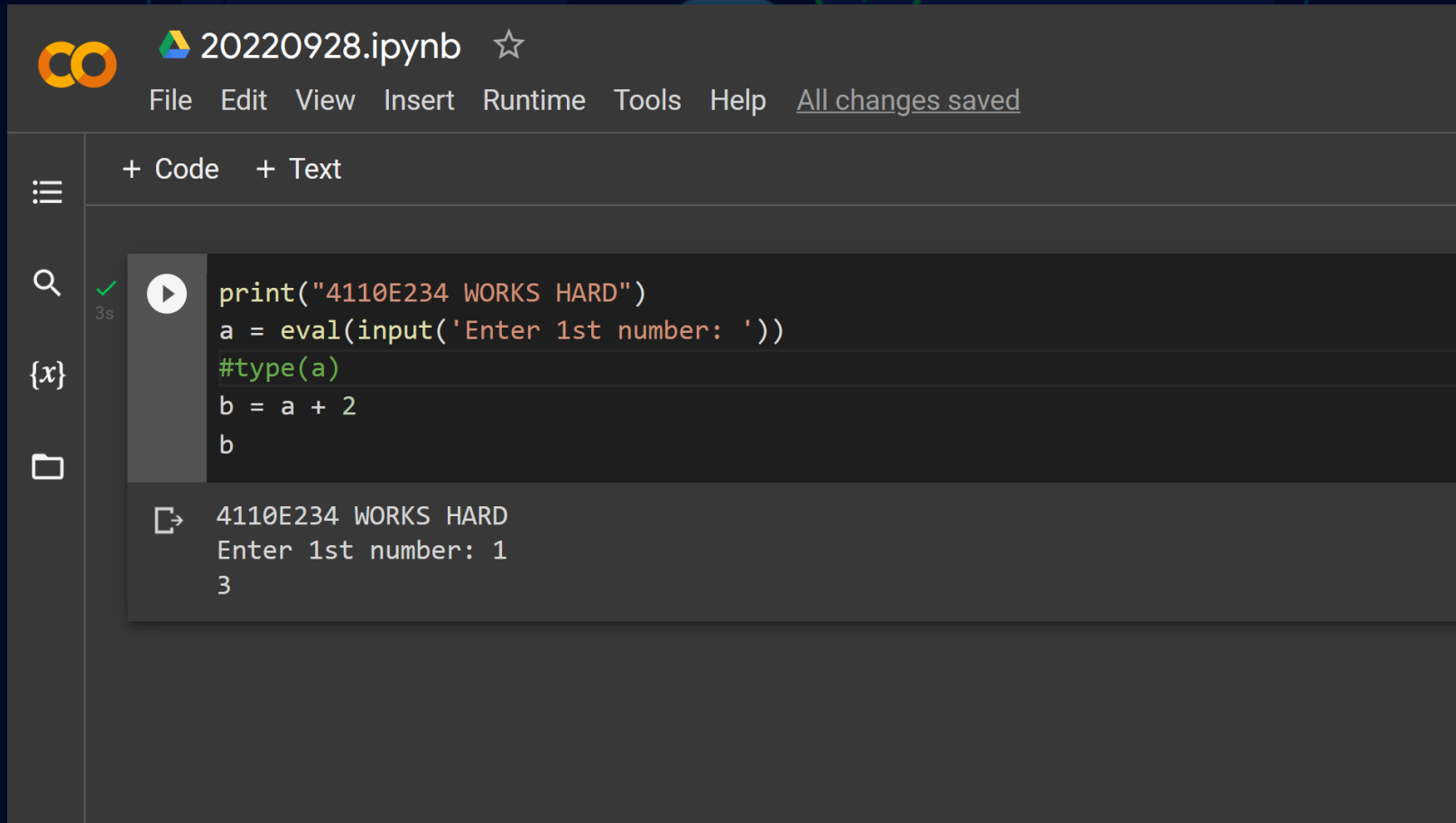
```
print("4110E234 WORKS")
input('Enter 1st number: ')
#type(a)
b = a + 2
b
```

Below the code cell, the output is displayed:

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



# Answer: 3



The image shows a JupyterLab interface with a dark theme. At the top, the file name '20220928.ipynb' is displayed with a star icon. Below the file name is a menu bar with options: File, Edit, View, Insert, Runtime, Tools, Help, and a link 'All changes saved'. On the left side, there is a sidebar with icons for a menu, search, variables, and a file explorer. The main area shows a code cell with a play button icon and a green checkmark. The code in the cell is:

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

Below the code cell, the output is displayed:

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

The background of the slide is a dark blue gradient. In the center, there is a faint, light blue Python logo. To the right of the logo, there is a green circle with a white checkmark. Below the logo, there is a faint bar chart with several bars of increasing height. The title 'PYTHON DATA TYPES' is written in white, uppercase letters, centered within a yellow rounded rectangle.

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

# Python Data



```
print("4110E234")
```

```
x = range(6)
```

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

```
#display x:
```

```
print(x)
```

```
#display the data 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

The image features a dark blue background with a faint, stylized illustration of a computer monitor. On the monitor's screen, there is a large, semi-transparent Python logo. Overlaid on the logo is a yellow rounded rectangle containing the text 'PYTHON OPERATORS' in white, bold, uppercase letters. Below this rectangle, the date '20220928' is displayed in a smaller white font. The background of the monitor screen also shows faint outlines of a code editor with lines of text and a bar chart at the bottom right.

# 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



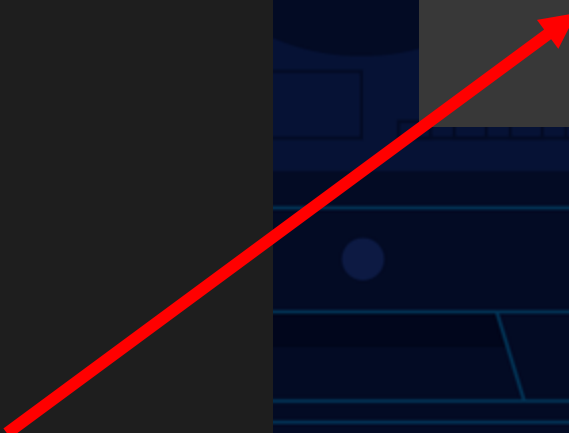
```
print("4110E234")
```

```
x = 5
```

```
y = 3
```

```
print(x >= y)
```

```
↳ 4110E234  
True
```

A red arrow originates from the expression `x >= y` in the code block and points to the `True` output in the result block.

# Python Comparison Operators

`z = x`

`["apple", "banana"]`

`["apple", "banana"]`

`z`

`x`

`y`

`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
```

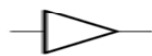
```
False
```

```
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



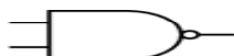
INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	1

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 Bitwise Operators

```
x = 5
y = 3

print(x & y)
```

1

binary  
 $1*2^2 + 0*2^1 + 1*2^0$

- $x = 5$   $(101)_2$
- $y = 3$   $(011)_2$
- $\text{print}(x \& y)$   $(001)_2$

# Python Bitwise 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>

