

# Variables and Data Types

---

Distributed Network and Systems Laboratory,  
Chonnam National University  
Presenter : Shivani Kolekar



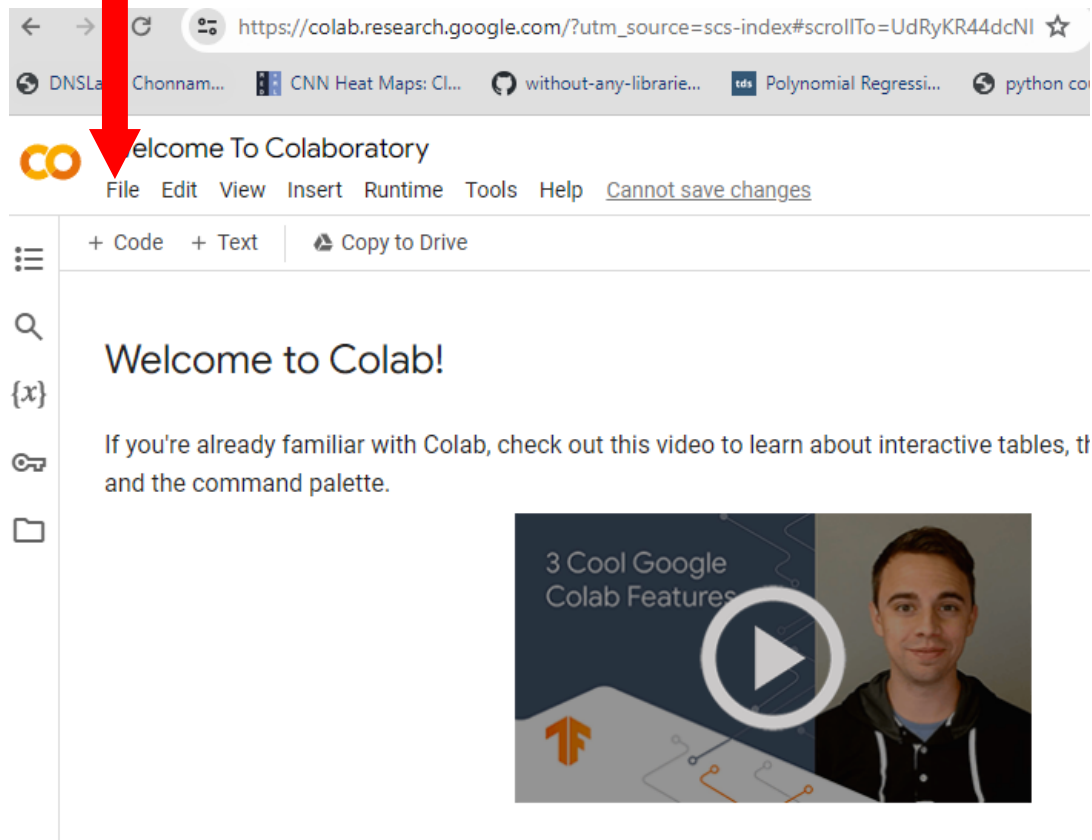
## Alternative to Pycharm – Google Colab

- ✓ Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more.
- ✓ they are stored in your Google Drive account.
- ✓ Easy sharing.

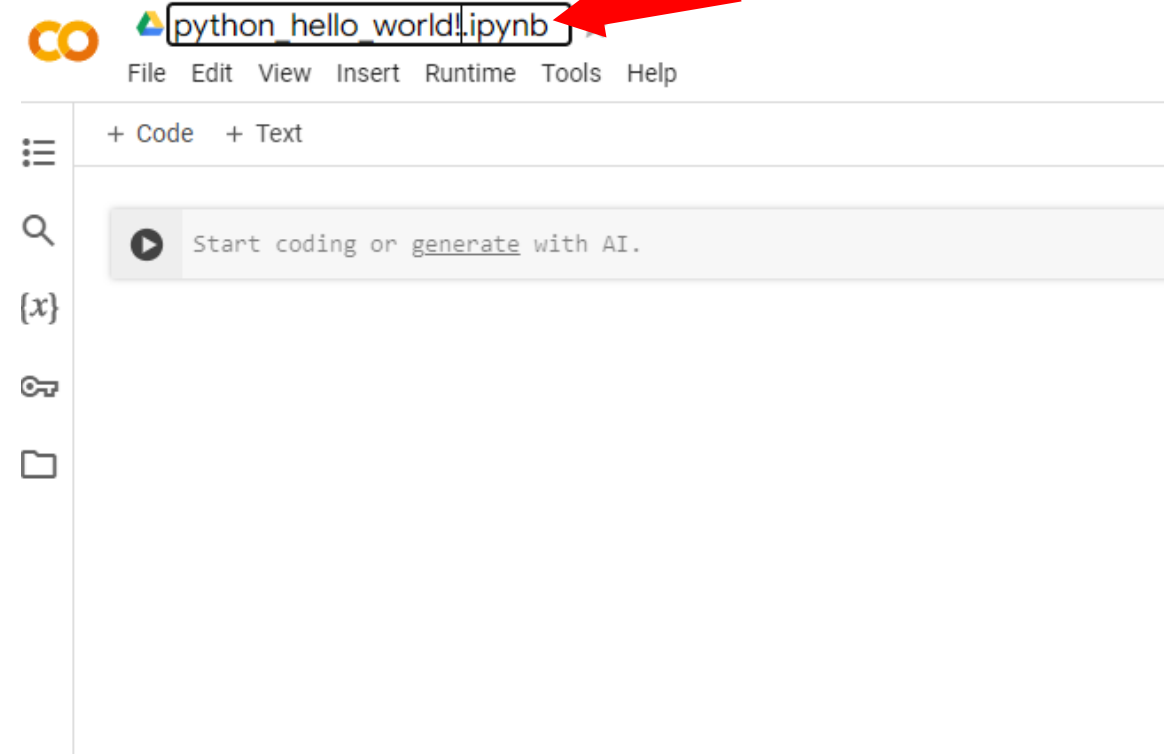
[https://colab.research.google.com/?utm\\_source=scs-index](https://colab.research.google.com/?utm_source=scs-index)

# Alternative to Pycharm – Google Colab

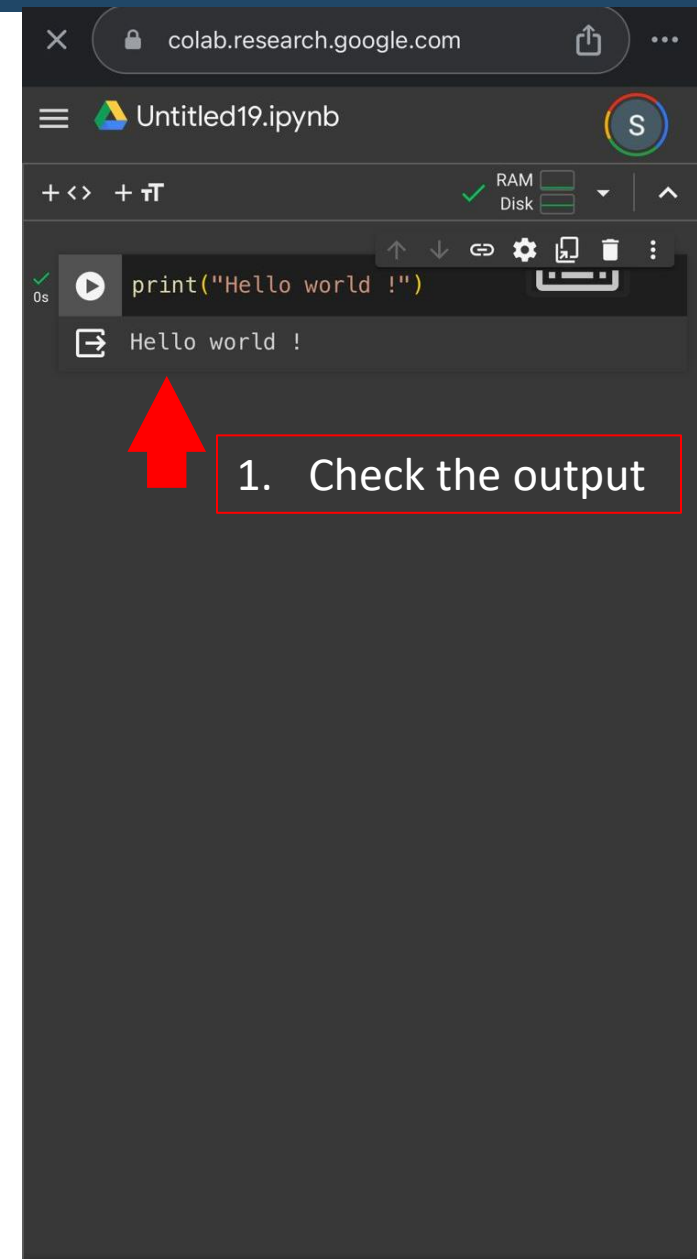
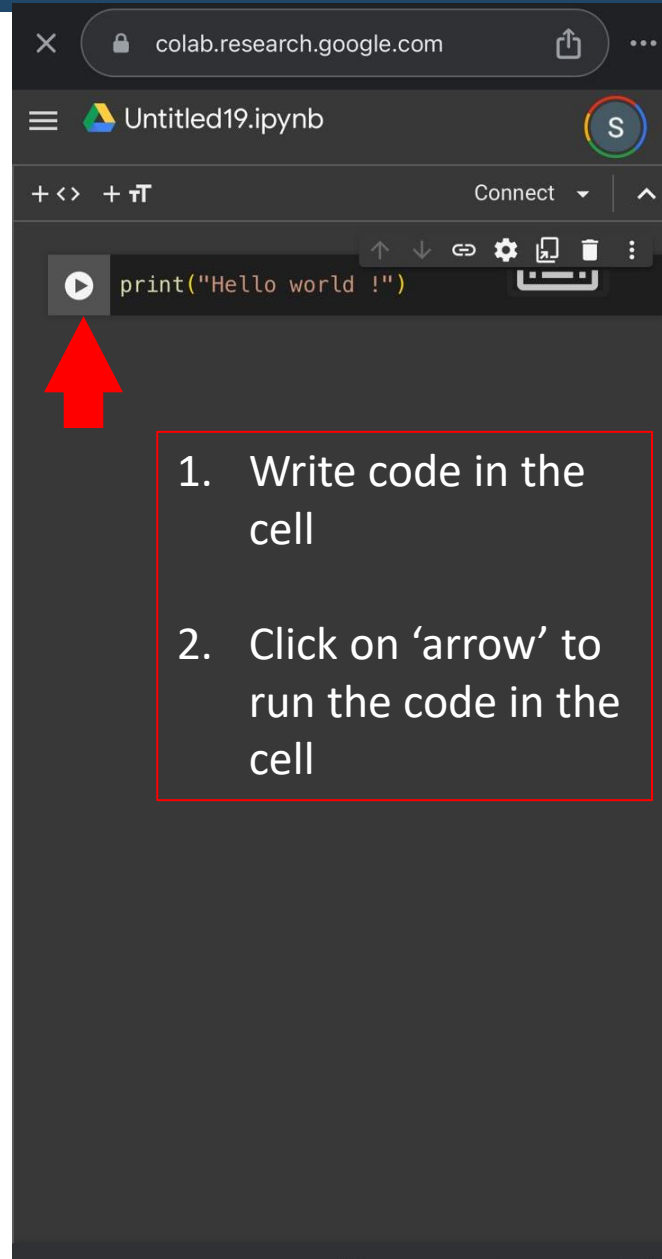
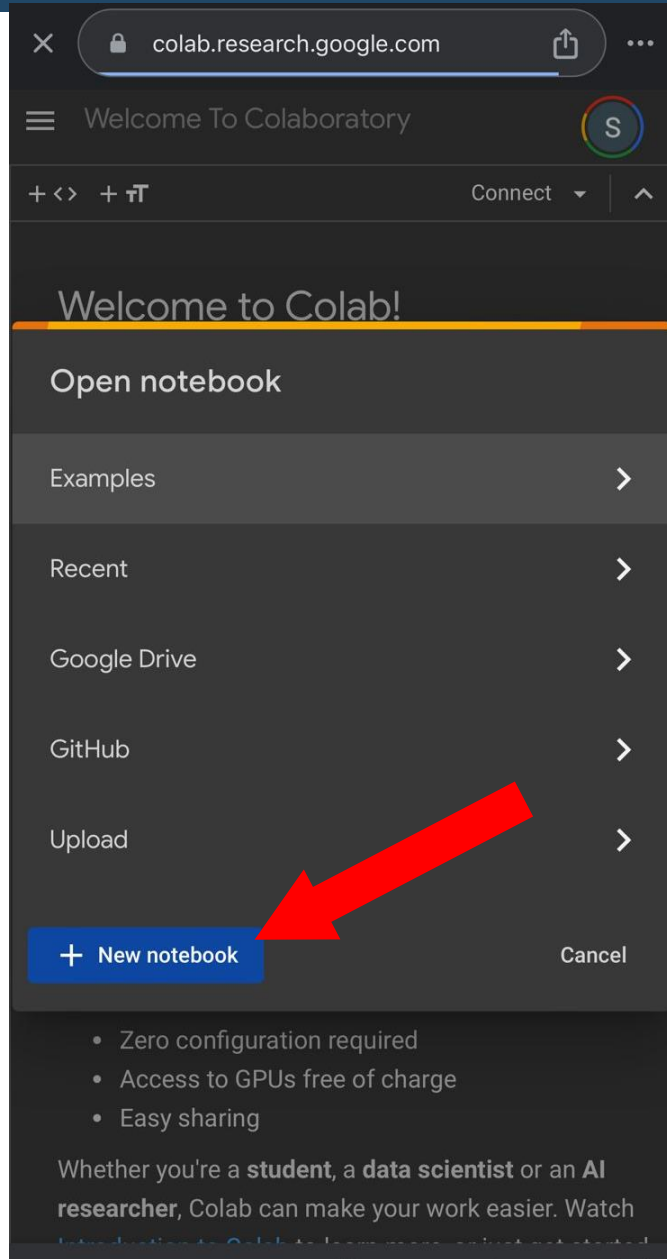
1. Go to 'File'
2. Select 'New notebook' option



3. Rename the newly created notebook



# Google Colab – for phone



- ✓ What is Python ?
- ✓ Python and AI
- ✓ Python Applications
- ✓ Compilers
- ✓ Python Interpreter
- ✓ Getting Python
- ✓ Hello World!

## *Today's goals*

- ✓ Computer memory
- ✓ Error and its types
- ✓ Understanding Variables and Data Types
- ✓ Basic Arithmetic Operations
- ✓ Q & A

# Computer memory

- ✓ 1 bit -> a single 0 or 1 (binary)
- ✓ 1 byte = 8 bits
- ✓ A typical laptop or desktop circa 2023
- ✓ ... has 4 to 32 Gigabytes of RAM, also known as main memory.
- ✓ 1 Gigabyte -> 1 billion bytes
- ✓ The programs that are running store their instructions and data (typically) in the RAM
- ✓ ... have 100s of Gigabytes up to several Terabytes (trillions of bytes) in secondary storage
  - . Long term storage of data, files
- ✓ Typically spinning disks (Hard disk drives - HDD) or solid state drives (SSD).



# Strings and string literals

1. Pieces of data are sequences of characters.
2. In programming terms, a sequence of characters that is used as data is called a string. When a string appears in the actual code of a program, it is called a string literal.
3. In Python code, string literals must be enclosed in quote marks.
4. The quote marks → string data begins and ends.

## Program 2-1 (output.py)

```
1 print('Kate Austen')
2 print('123 Full Circle Drive')
3 print('Asheville, NC 28899')
```

### Program Output

```
Kate Austen
123 Full Circle Drive
Asheville, NC 28899
```

## Program 2-2 (double\_quotes.py)

```
1 print("Kate Austen")
2 print("123 Full Circle Drive")
3 print("Asheville, NC 28899")
```

### Program Output

```
Kate Austen
123 Full Circle Drive
Asheville, NC 28899
```



# Strings and string literals

1. If you want a string literal to contain either a single-quote ( `'` ) or an apostrophe ( `'` ) as part of the string, you can enclose the string literal in double-quote marks.

## **Program 2-3** (apostrophe.py)

```
1 print("Don't fear!")  
2 print("I'm here!")
```

## **Program Output**

```
Don't fear!  
I'm here!
```

# Comments

1. Comments are short notes placed in different parts of a program.
2. Although comments are a critical part of a program, they are ignored by the Python interpreter.

```
1 # This program displays a person's  
2 # name and address.  
3 print('Kate Austen')  
4 print('123 Full Circle Drive')  
5 print('Asheville, NC 28899')
```

## Program Output

```
Kate Austen  
123 Full Circle Drive  
Asheville, NC 28899
```

# Comments in code

1. What are other ways to write a comment ?

1) `# comment`

2) `''' comment '''`

3) `"""  
 comment  
 """`

```
1 # This program displays a person's  
2 # name and address.  
3 print('Kate Austen')  
4 print('123 Full Circle Drive')  
5 print('Asheville, NC 28899')
```

## Program Output

```
Kate Austen  
123 Full Circle Drive  
Asheville, NC 28899
```

## *Try to solve:*

1. Try adding a comment – [This is a comment for print statement]
2. Write a statement that displays your name.
3. Write a statement that displays the following text:  
Python's the best!
4. Write a statement that displays the following text:  
The cat said "meow."

# Variables

# Assignment Statements

An assignment in Python has form:

**<variable> = <expression>**

- This means that variable is assigned value. i.e., after the assignment, variable "contains" value
- The equals sign (=) is NOT algebraic equality.
- It causes an action! The expression on the right is evaluated and the result is assigned to the variable on the left.


# Creating Variables

- ✓ Variables are memory blocks for storing data values.
- ✓ It is created the moment you first assign a value to it.


```
x = 17      # Defines and initializes x
```

- **Python is case-sensitive**

Code :

```
 a = 4  
A = "Sally"  
print(a)  
print(A)
```

Output :

```
 4  
Sally
```

# Python Data types

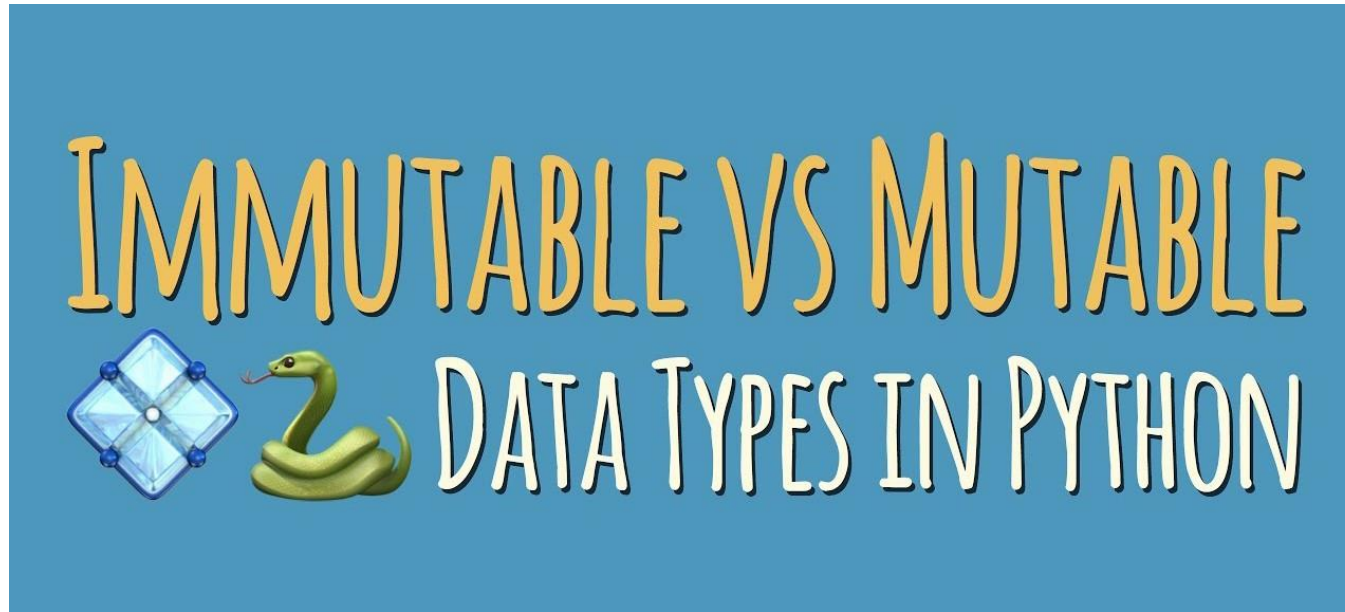
- ✓ **Strings (str):** Sequence of characters, enclosed in quotes.
- ✓ Example: name = "John"
- ✓ **Integers (int):** Whole numbers.
- ✓ Example: age = 15
- ✓ **Floating-Point Numbers (float):** Numbers with a decimal point.
- ✓ Example: temperature = 98.6
- ✓ **Booleans (bool):** Represents truth values: True or False.
- ✓ Example: is\_student = True



# Python Data types

A **data type** is a categorization of values.

Data Type	Description	Example
int	integer. An immutable number of unlimited magnitude	42
float	A real number. An immutable floating point number, system defined precision	3.1415927
str	string. An immutable sequence of characters	'Wikipedia'
bool	boolean. An immutable truth value	True, False
tuple	Immutable sequence of mixed types.	(4.0, 'UT', True)
list	Mutable sequence of mixed types.	[12, 3, 12, 7, 6]
set	Mutable, unordered collection, no duplicates	{12, 6, 3}
dict	dictionary a.k.a. maps, A mutable group of (key, value pairs)	{'k1': 2.5, 'k2': 5}



- ✓ An **immutable** value is one that cannot be changed by the programmer after you create it; e.g., numbers, strings, etc.
- ✓ A **mutable** values is one that can be changed; e.g., sets, lists, etc.

# The type function

```
>>> x = 17
>>> type(x)
<class 'int'>
>>> y = -20.9
>>> type(y)
<class 'float'>
>>> type(w)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'w' is not defined
>>> lst = [1, 2, 3]
>>> type(lst)
<class 'list'>
>>> type(20)
<class 'int'>
>>> type( (2, 2.3) )
<class 'tuple'>
>>> type('abc')
<class 'str'>
>>> type( {1, 2, 3} )
<class 'set'>
>>> type(print)
<class 'builtin_function_or_method'>
```

- *Class* is another name for data type.
- Data type is a categorization.
- "What kind of thing is the value this variable refers to?"

# Naming Variables

## Below are (most of) the rules for naming variables:

- ✓ Variable names must begin with a letter or underscore (\_) character.
- ✓ After that, use any number of letters, underscores, or digits.
- ✓ Case matters: "score" is a different variable than "Score."
- ✓ You can't use reserved words; these have a special meaning to Python and cannot be variable names.

# Python Keywords

- **Python Reserved Words.**
- **Also known as Keywords**
- ✓ and, as, assert, break, class, continue, def, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, nonlocal, None, not, or, pass, raise, return, True, try, while, with, yield, del

# Arithmetic Operations

- Here are some useful operations you can perform on numeric data types.

Name	Meaning	Example	Result
+	Addition	$34 + 1$	35
-	Subtraction	$34.0 - 0.1$	33.9
*	Multiplication	$300 * 30$	9000
/	Float division	$1 / 2$	0.5
//	floor division	$1 // 2$	0
**	Exponentiation	$4 ** 0.5$	2.0
%	Remainder	$20 \% 3$	2

$(x \% y)$  is often referred to as "x mod y"

## Activity

Calculate Body Mass Index (BMI) of a person where his mass(weight) and height is given as follows,

mass = 30 kg

Height = 1.5m

BMI ?

$$\text{Formula : BMI} = \text{mass}/\text{height}^2$$

**Let's Code!**



# Errors : Syntax error

- ✓ We will encounter three types of errors when developing our Python program.
1. **syntax errors:** these are ill-formed Python and caught by the interpreter prior to executing your code.

```
>>> 3 = x
      File "<stdin>", line 1
      SyntaxError: can't assign to
      literal
```

These are typically the easiest to find and fix.

# Errors : Runtime error

- ✓ **runtime errors:** you try something illegal while your code is executing

```
>>> x = 0
>>> y = 3
>>> y / x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

- ✓ **logic errors:** your program runs but returns an incorrect result

```
def calculate_average(num1, num2, num3):  
    average = (num1 + num2 + num3) / 2 # Logic error here  
    return average  
  
# Testing the function  
result = calculate_average(10, 20, 30)  
print("The average is:", result)
```

This program is syntactically fine and runs without error. But it probably doesn't do what the programmer intended.

How would you fix it?

**Logic errors are often the hardest errors to find and fix.**

# Thank You