

<https://www.youtube.com/watch?v=5aYvPsP2zvw>

<https://www.youtube.com/watch?v=CXFwenalEi8>

The screenshot shows a Jupyter Notebook window with a sidebar on the left containing icons for a file explorer, search, and code editor. The main area displays a notebook titled "Numbers" with the text "Integers and floats work as you would expect from other languages:". Below this, there are three code cells. The first cell, labeled [1], contains the code `x = 3` and `print(x, type(x))`, with the output `3 <class 'int'>`. The second cell, labeled [2], contains the code `print(x + 1)` (commented "# Addition"), `print(x - 1)` (commented "# Subtraction"), `print(x * 2)` (commented "# Multiplication"), and `print(x ** 2)` (commented "# Exponentiation"). The output for this cell is a vertical list of numbers: 4, 2, 6, and 9. The third cell, labeled [3], contains the code `x += 1`, `print(x)`, `x *= 2`, and `print(x)`. The output for this cell is a vertical list of numbers: 4 and 8. At the bottom of the notebook window, there is a status bar showing a green checkmark and "0s". Below the notebook window is a Windows taskbar with the Start button, a search bar containing the text "Type here to search", and several application icons including the Task View button, File Explorer, and Microsoft Edge.

```
[1] x = 3
    print(x, type(x))

3 <class 'int'>
```

```
[2] print(x + 1) # Addition
    print(x - 1) # Subtraction
    print(x * 2) # Multiplication
    print(x ** 2) # Exponentiation

4
2
6
9
```

```
[3] x += 1
    print(x)
    x *= 2
    print(x)

4
8
```

0s



▼ Booleans



Python implements all of the usual operators for Boolean logic, but uses English words rather than symbols (&&, ||, etc.):



```
[5] t, f = True, False  
    print(type(t))
```

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

Now we let's look at the operations:

```
[6] print(t and f) # Logical AND;  
    print(t or f)  # Logical OR;  
    print(not t)   # Logical NOT;  
    print(t != f)  # Logical XOR;
```

```
False  
True  
False  
True
```

▼ Strings



✓ 0s completed at 12:56 PM



🔍 Type here to search





▼ Strings



```
[7] hello = 'hello' # String literals can use single quotes
    world = "world" # or double quotes; it does not matter
    print(hello, len(hello))
```

```
hello 5
```

```
[8] hw = hello + ' ' + world # String concatenation
    print(hw)
```

```
hello world
```

```
[9] hw12 = '{} {} {}'.format(hello, world, 12) # string formatting
    print(hw12)
```

```
hello world 12
```

String objects have a bunch of useful methods; for example:



```
[10] s = "hello"
     print(s.capitalize()) # Capitalize a string
     print(s.upper())      # Convert a string to uppercase; prints "HELLO"
     print(s.rjust(7))     # Right-justify a string, padding with spaces
```

✓ 0s completed at 12:5



🔍 Type here to search




```
[ ] def sign(x):  
    if x > 0:  
        return 'positive'  
    elif x < 0:  
        return 'negative'  
    else:  
        return 'zero'  
  
    for x in [-1, 0, 1]:  
        print(sign(x))
```

negative	
zero	
positive	

We will often define functions to take optional keyword arguments, like this:

```
[ ] def hello(name, loud=False):
    if loud:
        print('HELLO, {}'.format(name.upper()))
    else:
        print('Hello, {}'.format(name))

hello('Bob')
hello('Fred', loud=True)
```

✓ 0s completed at 12:56 PM



<https://www.kaggle.com/mauriciofigueiredo/fruit-classification-with-a-simple-cnn>