GOVERNMENT OF
WESTERN AUSTRALIA

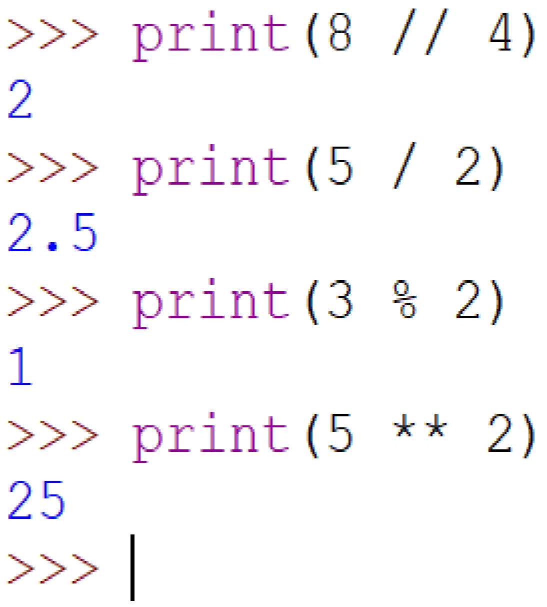
TAFE WA South Metropolitan

User Input and Operators

(Session 3)

Review (last week)

- Arithmetical Operators
  - *integer division* uses a double-slash sign **//**
    - for instance: **print(8 // 4)** # output is 2
  - *float (regular) division* uses a single-slash sign **/**
    - for instance: **print(5 / 2)** #output is 2.5
  - Modulus operator uses the symbol **%**
    - for instance: **print(3 % 2)** #output is 1
  - Exponent operator uses the symbol ******
    - For instance: **print(5 ** 2)** #output is 25



```
>>> print(8 // 4)
2
>>> print(5 / 2)
2.5
>>> print(3 % 2)
1
>>> print(5 ** 2)
25
>>> 
```

Review (last week)

- *Integer*
  - is a whole number, positive or negative, without decimals
  - for instance, 1, -5, 100, -50, -5000
- Floating point or *Float*
  - is a number, positive or negative, containing decimals
  - for instance, 1.10, -5.20, 100.995, -5000.29
- *Strings* – surrounded by single or double quotation marks
  - used when you need to process text
  - For instance, ‘Hello World!’, “I am a string”, “ABC”, “abcd”
- Boolean Values or *Booleans*
  - Represent one of two values: True or False

Activity 1: What types of literals (numbers or string) are the following five examples

- 3.0
- “703”
- -300
- 100000
- “Python 123”

Activity 2: Assignment Operators

```
# Augmented arithmetic assignments. What does +=, -=, *= and //= stand for?
#Practice makes perfect…
# Experiment with the following Python code
x = 2   # assigns value of 2 into x
x += 4  # the same as x = x + 4
x -= 4  # the same as x = x - 4
x *= 3  # the same as x = x * 3
x //= 3  # the same as x = x // 3
print(x)
```

Overview Week 3

- Getting the Data Types
- Type Conversion
- Using **input()** function
- Python Comparison Operators
- Python Logical Operators
- Python Bitwise Operators

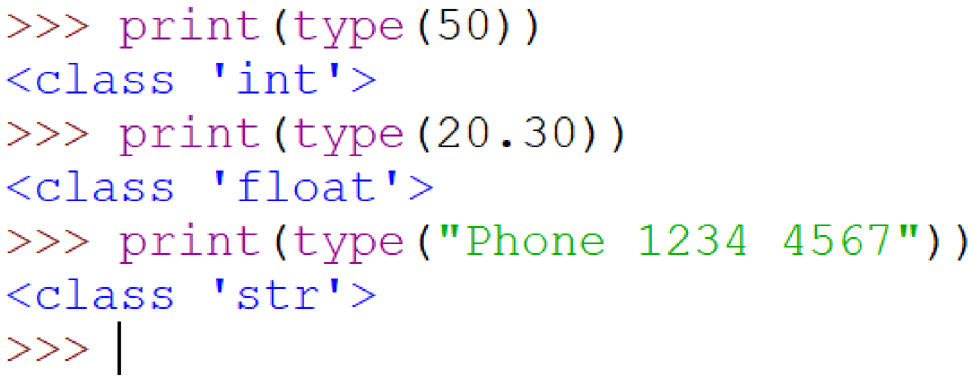
Getting the Data Type

- You can get the data type using the **type()** function
  - For instance:

print(type(50)) # outputs class ‘int’>

print(type(20.30)) # outputs class ‘float’>

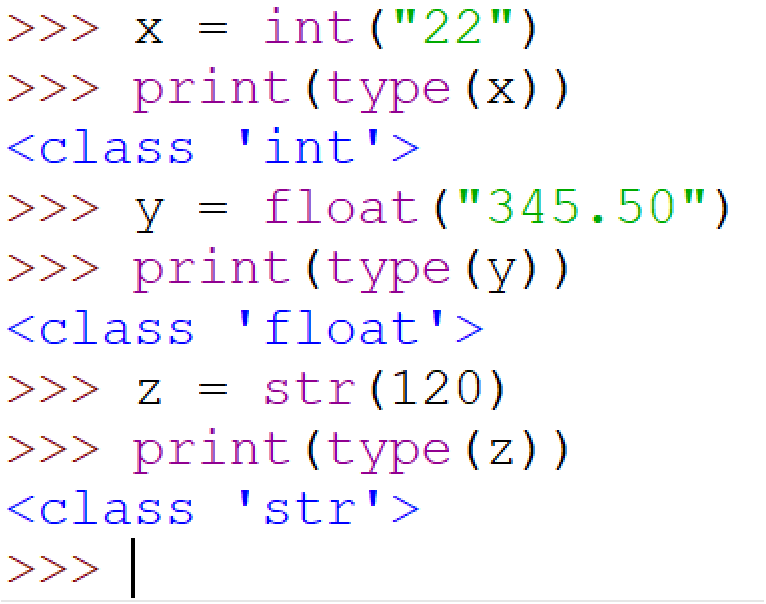
print(type("Phone: 12345 4567")) # outputs class ‘str’>



```
>>> print(type(50))
<class 'int'>
>>> print(type(20.30))
<class 'float'>
>>> print(type("Phone 1234 4567"))
<class 'str'>
>>> 
```

Type Conversion (Type Casting)

- You can convert from one type to another with int(), float() and str() functions
- The **int()** function takes one argument and tries to convert it into an integer
  - For instance, x = int("22")
- The **float()** function takes one argument and tries to convert into a float
  - For instance, y = float("345.50")
- The **str()** function converts numbers to string
  - For instance, z = str(120)



```
>>> x = int("22")
>>> print(type(x))
<class 'int'>
>>> y = float("345.50")
>>> print(type(y))
<class 'float'>
>>> z = str(120)
>>> print(type(z))
<class 'str'>
>>> 
```

The input() function

- Built-in function that reads data entered by users
- Syntax*:* ***input(prompt)***
    - prompt (optional) – it is a default message before input
    - returns a string value
- Python stops executing and reads data from the user
- It continues when the user has given some input

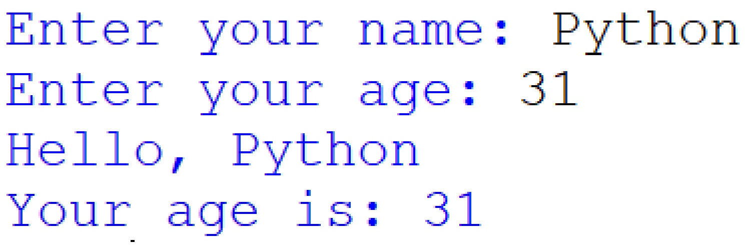
User Input

- The following example prints user input
- The input () function takes the input from the user
- Input is converted into a string

```
# get input from user
print("Enter username: ", end="")
user_input = input()
print("Username:", user_input)
```

Activity 3: How input() works in Python?

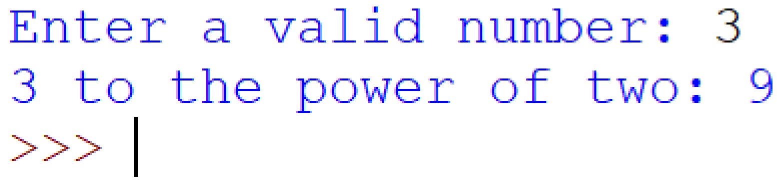
```
# this program asks for the user's name and age, and print it
print("Enter your name: ", end='')
input_name = input()
print("Enter your age: ", end='')
input_age = input()
print("Hello,", input_name)
print("Your age is", input_age)

#short version
input_name = input("Enter your name: ")
input_age = input("Enter your age: ")
print("Hello,", input_name, "\nYour age is:", input_age)
```



```
Enter your name: Python
Enter your age: 31
Hello, Python
Your age is: 31
```

Activity 4: Type casting and input()

```
# this program calculates the power of two
print("Enter a valid number: ", end="")  # to avoid Error
input_num = int(input())  # converts string to integer
power_two = input_num ** 2
print(input_num, "to the power of two:", power_two)
```

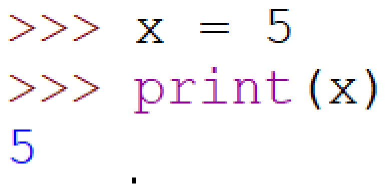


```
Enter a valid number: 3
3 to the power of two: 9
>>> 
```

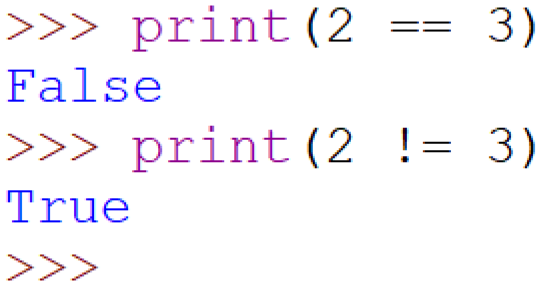
```
# short version
input_num = int(input("Enter a valid number: "))
print(input_num, "to the power of two:", input_num ** 2)
```

Assignment and Comparison Operators

- **Assignment** operators are used to assign values to variable.
  - For example: x = 5
- **Comparison** operators are used to compare two values
  - Equality operators use the **==** operator. Are two values equal? If not, the result of comparison is False.
  - For example: print(2 == 3)   # output is False
  - Not equal operator !=. If two values are not equal, the result of comparison is True
  - For example, print(2 != 3)   #output is True



```
>>> x = 5
>>> print(x)
5
```



```
>>> print(2 == 3)
False
>>> print(2 != 3)
True
>>>
```

Python Comparison Operators

Operator	Name	Example	Output example

==	Equal	print(3 == 3)
		print(3 == 4)
			True
			False

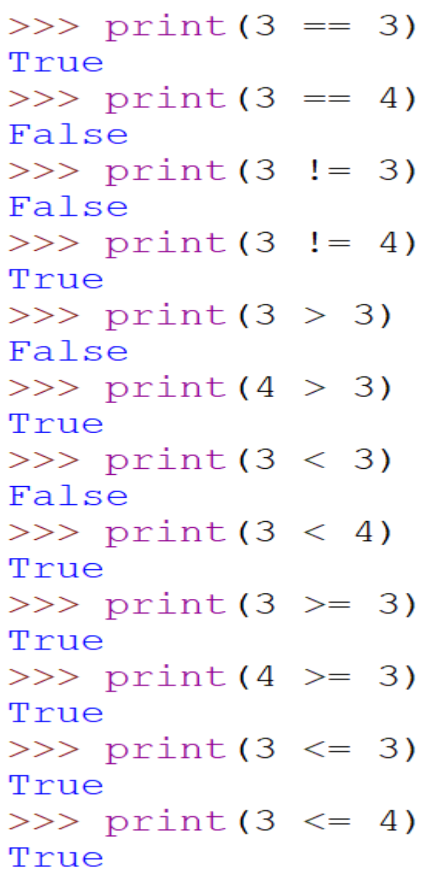
!=	Not equal	print(3 != 3)
		print(3 != 4)
			False
			True

>	Greater than	print(3 > 3)
		print(4 > 3)
			False
			True

<	Less than	print(3 < 3)
		print(3 < 4)
			False
			True

>=	Greater than or equal to	print(3 >= 3)
		Print(4 >= 3)
			True
			True

<=	Less than or equal to	print(3 <= 3)
		Print(3 <= 4)
			True
			True



```
>>> print(3 == 3)
True
>>> print(3 == 4)
False
>>> print(3 != 3)
False
>>> print(3 != 4)
True
>>> print(3 > 3)
False
>>> print(4 > 3)
True
>>> print(3 < 3)
False
>>> print(3 < 4)
True
>>> print(3 >= 3)
True
>>> print(4 >= 3)
True
>>> print(3 <= 3)
True
>>> print(3 <= 4)
True
```

Activity 5: Comparison Operators

- What output will you get with the following:

```
x = 3
y = 1
z = 1
print(x > z)
print(x == z)
print(y != z)
print(y <= z)
print(x > y > z)
```

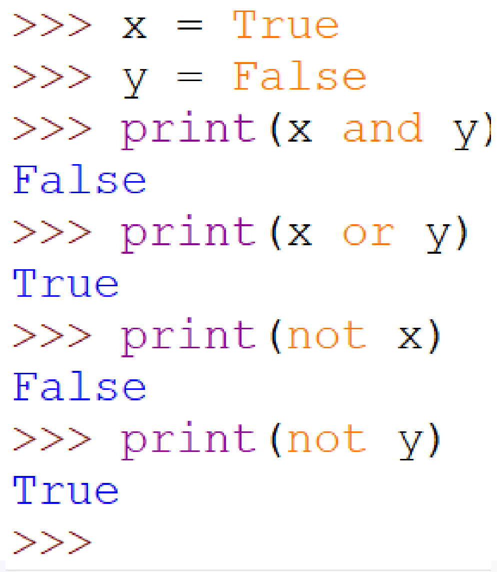
Python Logical Operators

- Logical operators are **and**, **or**, **not** operators

x = True

y = False

Operator	Name	Example	Output
and	True if both operands are true	print(x and y)	False
or	True if either of the operands is true	print(x or y)	True
not	True if operand is false	not x not y	False True



```
>>> x = True
>>> y = False
>>> print(x and y)
False
>>> print(x or y)
True
>>> print(not x)
False
>>> print(not y)
True
>>>
```

Activity 6: Logical Operators

- What output will you get with the following:

```
x = True
y = False
print(x and y)
print(x or y)
print(not x)
print(not y and x > y)
print(not(not x))
```

Converting int to binary

- To reveal the bits making up an integer number we can use **bin()** function, binary values are represented by prefix 0b

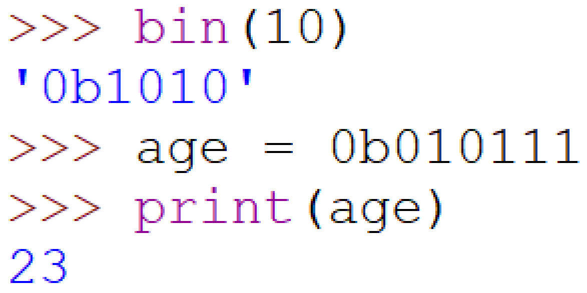
# the bin() function takes an integer and returns a binary string

bin (10)   # returns a string which starts with prefix 0b

# print() automatically converts the binary to int

age = 0b010111

print(age)   # outputs 23



```
>>> bin(10)
'0b1010'
>>> age = 0b010111
>>> print(age)
23
```

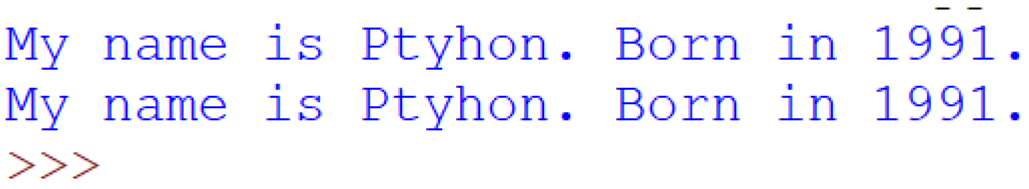
String Formatting in Python

- Using **str.format()**.
- The replacement fields are marked by curly braces
- Example:

```
name = "Ptyhon"

born = 1991

print("My name is {}. Born in {}.".format(name, born))
```

- Using **f-strings** (formatted string literals)

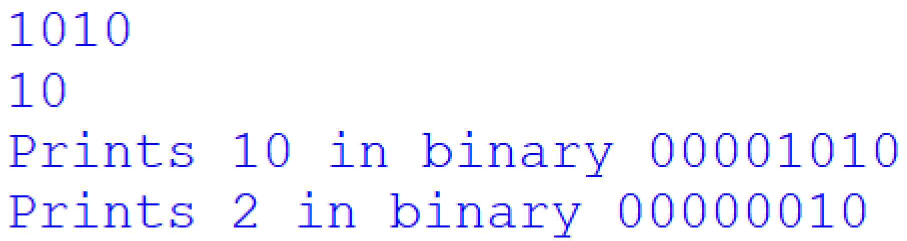
```
print(f"My name is {name}. Born in {born}.")
```



```
My name is Ptyhon. Born in 1991.
My name is Ptyhon. Born in 1991.
>>>
```

Using Bit Strings to display Binary sequence

```
# This program prints 10 and 2 in binary using a formatted string literal
print(f"{10:b}")  # integer to binary conversion, outputs 1010
print(f"{2:b}")   #integer to binary conversion, outputs 10
print(f"Prints 10 in binary {10:08b}")  # on 8 zero-padded digits
print(f"Prints 2 in binary {2:08b}")  # on 8 zero-padded digits
```

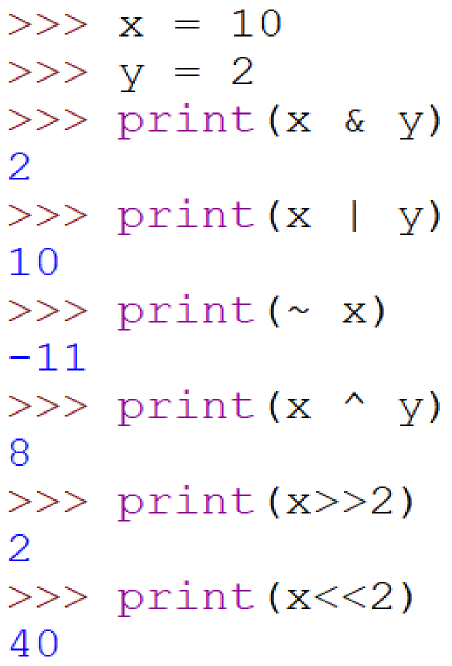


```
1010
10
Prints 10 in binary 00001010
Prints 2 in binary 00000010
```

Bitwise Operators – comparing binary numbers

- Let **x = 10** (0000 1010 in binary) and **y = 2** (0000 0010 in binary)

Operator	Name	Example	Binary	Output example
&	Bitwise AND	x & y	0000 1010 0000 0010	0000 0010 = 2
|	Bitwise OR	x | y	0000 1010 0000 0010	0000 1010 = 10
~	Bitwise NOT	~ x	0000 1010	1111 0101 = -11
^	Bitwise XOR	x ^ y	0000 1010 0000 0010	00001000 = 8
>>	Bitwise right shift	x>>2	0000 1010	0000 0010 = 2
<<	Bitwise left shift	x<<2	0000 1010	0010 1000 = 40
10 in Binary Value	0 0 0 0 **1** 0 **1** 0 128 64 32 16 **8** 4 **2** 1			

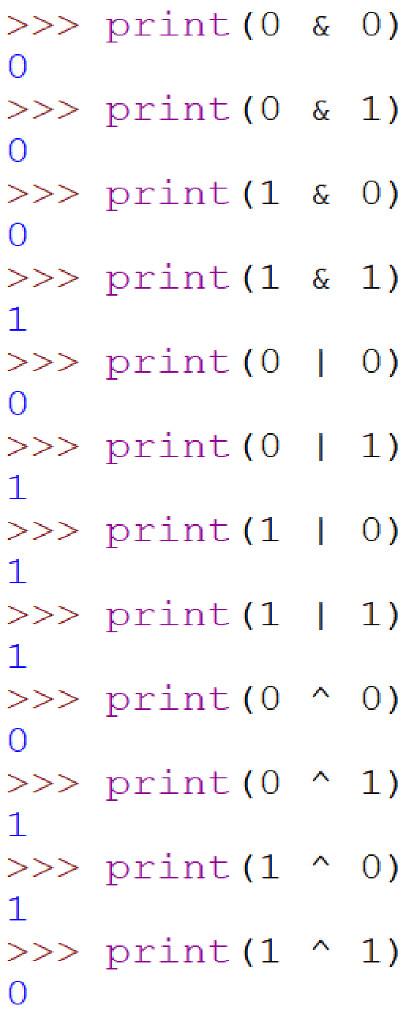


```
>>> x = 10
>>> y = 2
>>> print(x & y)
2
>>> print(x | y)
10
>>> print(~ x)
-11
>>> print(x ^ y)
8
>>> print(x>>2)
2
>>> print(x<<2)
40
```

Bitwise operations (&, |, ^)

- & - bitwise conjunction
- | bitwise disjunction
- ^ bitwise exclusive or

Argument x	Argument y	x & y	x | y	x ^ y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

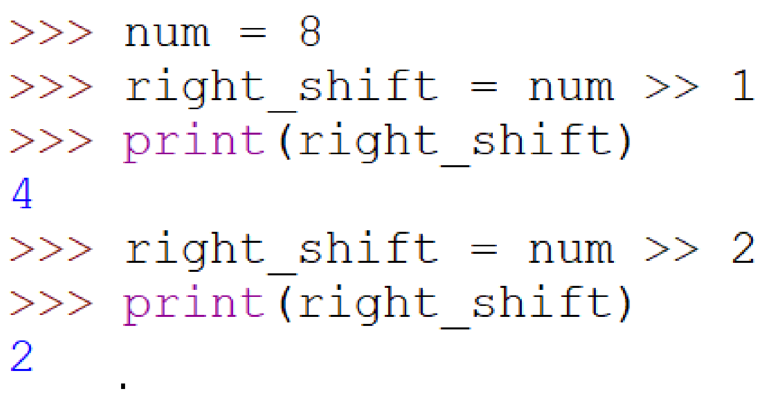


```
>>> print(0 & 0)
0
>>> print(0 & 1)
0
>>> print(1 & 0)
0
>>> print(1 & 1)
1
>>> print(0 | 0)
0
>>> print(0 | 1)
1
>>> print(1 | 0)
1
>>> print(1 | 1)
1
>>> print(0 ^ 0)
0
>>> print(0 ^ 1)
1
>>> print(1 ^ 0)
1
>>> print(1 ^ 1)
0
```

Activity 7: Binary right-shift

- the right shift operator is **>>**
- For instance, 8 >> 2
- The left argument is an integer value whose bits are shifted

```
num = 8
right_shift = num >> 1      # 8 // 2 = 4, the same as integer division by 2
print(right_shift)
right_shift = num >> 2     #8 // 2 = 4, and 4  // 2 = 2
print(right_shift)
```

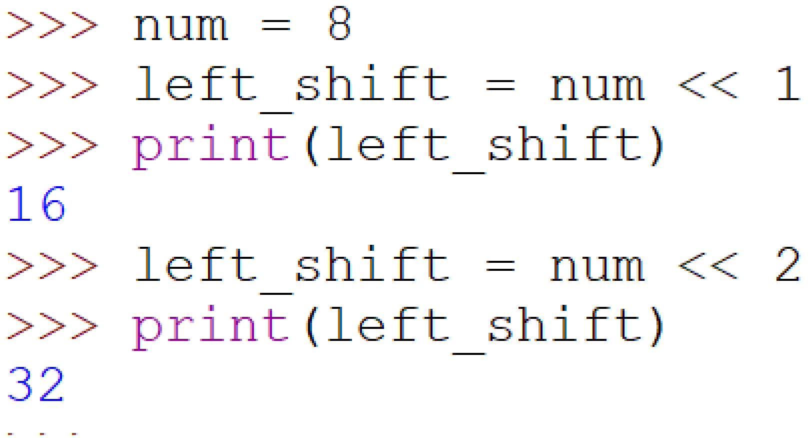


```
>>> num = 8
>>> right_shift = num >> 1
>>> print(right_shift)
4
>>> right_shift = num >> 2
>>> print(right_shift)
2
```

Activity 8: Binary left-shift

- The left shift operator is **<<**
- For instance, 8 << 2
- The left argument is an integer value whose bits are shifted

```
num = 8
left_shift = num << 1   # the same as integer multiplication by 2
print(left_shift)
left_shift = num << 2    # the same as integer multiplication by 4
print(left_shift)
```



```
>>> num = 8
>>> left_shift = num << 1
>>> print(left_shift)
16
>>> left_shift = num << 2
>>> print(left_shift)
32
```

Questions?