

# Expressions

# Numeric Expressions (1 of 2)

- Because of the lack of mathematical symbols on computer keyboards - we use “computer-speak” to express the classic math operations
- Asterisk is multiplication
- Exponentiation (raise to a power) looks different than in math

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

# Numeric Expressions (2 of 2)

```
>>> xx = 2
>>> xx = xx + 2
>>> print(xx)
4
>>> yy = 440 * 12
>>> print(yy)
5280
>>> zz = yy / 1000
>>> print(zz)
5.28
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3
>>> print(4 ** 3)
64
```

5

4 R 3

23

20

3

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

# Order of Evaluation

- When we string operators together - Python must know which one to do first
- This is called “operator precedence”
- Which operator “takes precedence” over the others?

**x** = 1 + 2 \* 3 - 4 / 5 \*\* 6

# Operator Precedence Rules

Highest precedence rule to lowest precedence rule:

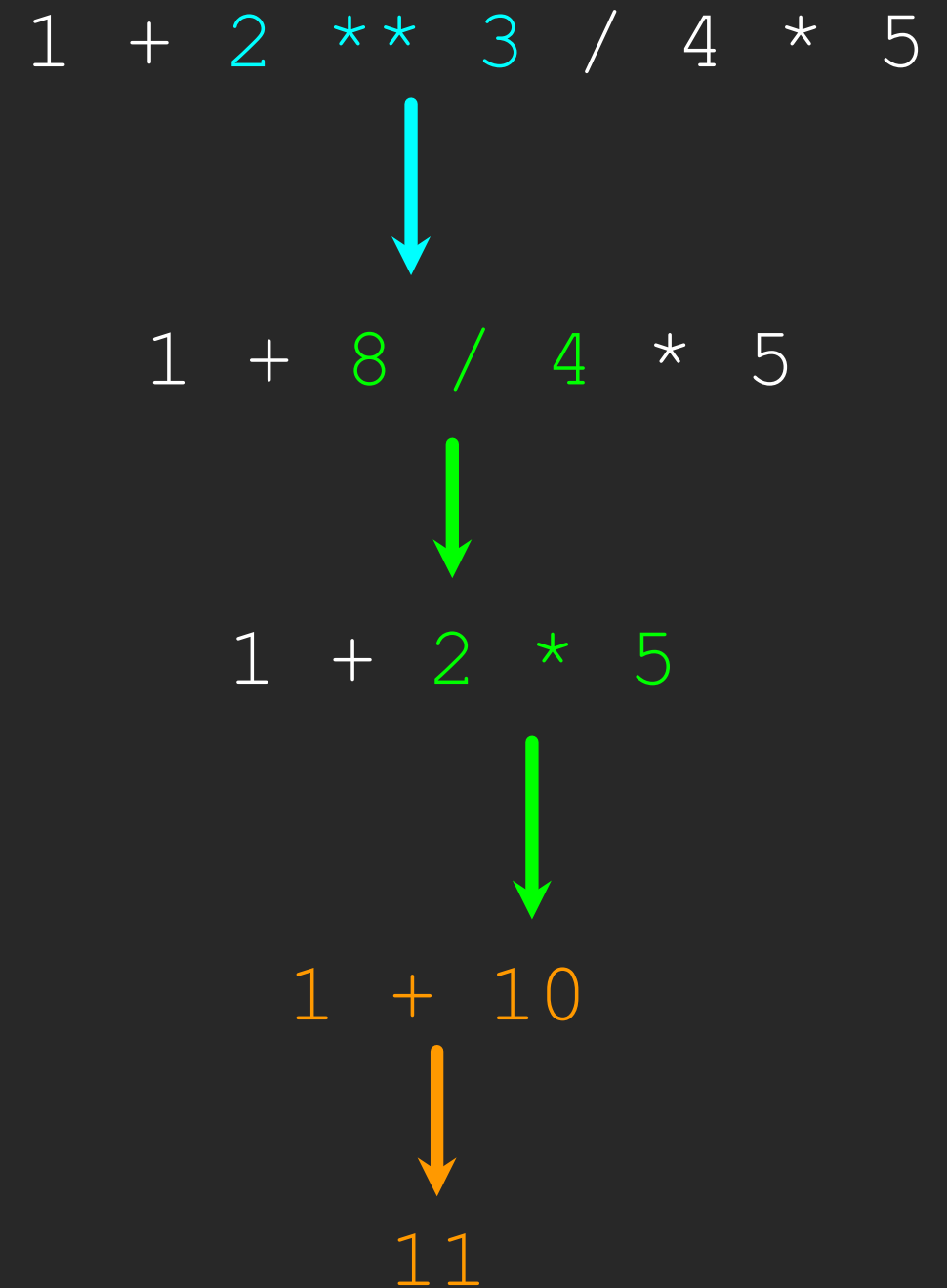

- Parentheses are always respected
- Exponentiation (raise to a power)
- Multiplication, Division, and Remainder
- Addition and Subtraction
- Left to right

Parenthesis  
Power  
Multiplication  
Addition  
Left to Right



```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```

Parenthesis  
Power  
Multiplication  
Addition  
Left to Right



# Operator Precedence

- Remember the rules top to bottom
- When writing code - use parentheses
- When writing code - keep mathematical expressions simple enough that they are easy to understand
- Break long series of mathematical operations up to make them more clear

Parenthesis  
Power  
Multiplication  
Addition  
Left to Right



# What does “Type” Mean?

- In Python variables, literals, and constants have a “**type**”
- Python knows the **difference** between an integer number and a string
- For example “**+**” means “addition” if something is a number and “concatenate” if something is a string

```
>>> ddd = 1 + 4
>>> print(ddd)
5
>>> eee = 'hello ' + 'there'
>>> print(eee)
hello there
```

**concatenate** = put together



# Type Matters

- Python knows what “**type**” everything is
- Some operations are prohibited
- You cannot “add 1” to a string
- We can ask Python what type something is by using the **type()** function

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>TypeError: Can't convert
'int' object to str implicitly
>>> type(eee)
<class 'str'>
>>> type('hello')
<class 'str'>
>>> type(1)
<class 'int'>
>>>
```

# Several Types of Numbers

- Numbers have two main types
  - **Integers** are whole numbers:  
-14, -2, 0, 1, 100, 401233
  - **Floating Point Numbers** have  
decimal parts: -2.5 , 0.0, 98.6, 14.0
- There are other number types - they  
are variations on float and integer

```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class 'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class 'float'>
>>>
```

# Type Conversions

- When you put an integer and floating point in an expression, the integer is **implicitly** converted to a float
- You can control this with the built-in functions `int()` and `float()`

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class 'float'>
>>>
```

# Integer Division

- Integer division produces a floating point result

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(10.0 / 2.0)
5.0
>>> print(99.0 / 100.0)
0.99
```

Integer division was different in Python 2.x

# String Conversions

- You can also use `int()` and `float()` to convert between strings and integers
- You will get an **error** if the string does not contain numeric characters

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object
to str implicitly
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
with base 10: 'x'
```

# User Input

- We can instruct Python to pause and read data from the user using the `input()` function
- The `input()` function returns a string

```
nam = input('Who are you?')  
print('Welcome', nam)
```

Who are you? **Chuck**  
Welcome Chuck



# Making a Program