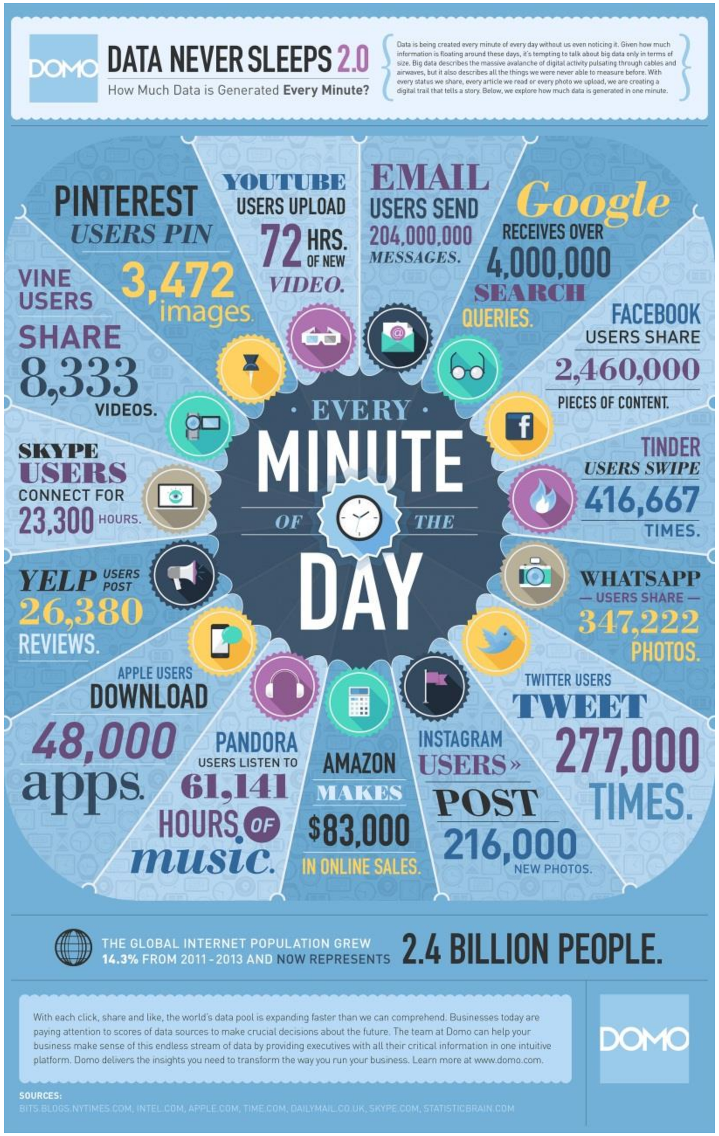
Data and Expressions

Data deluge



DATA NEVER SLEEPS 2.0

How Much Data is Generated **Every Minute?**

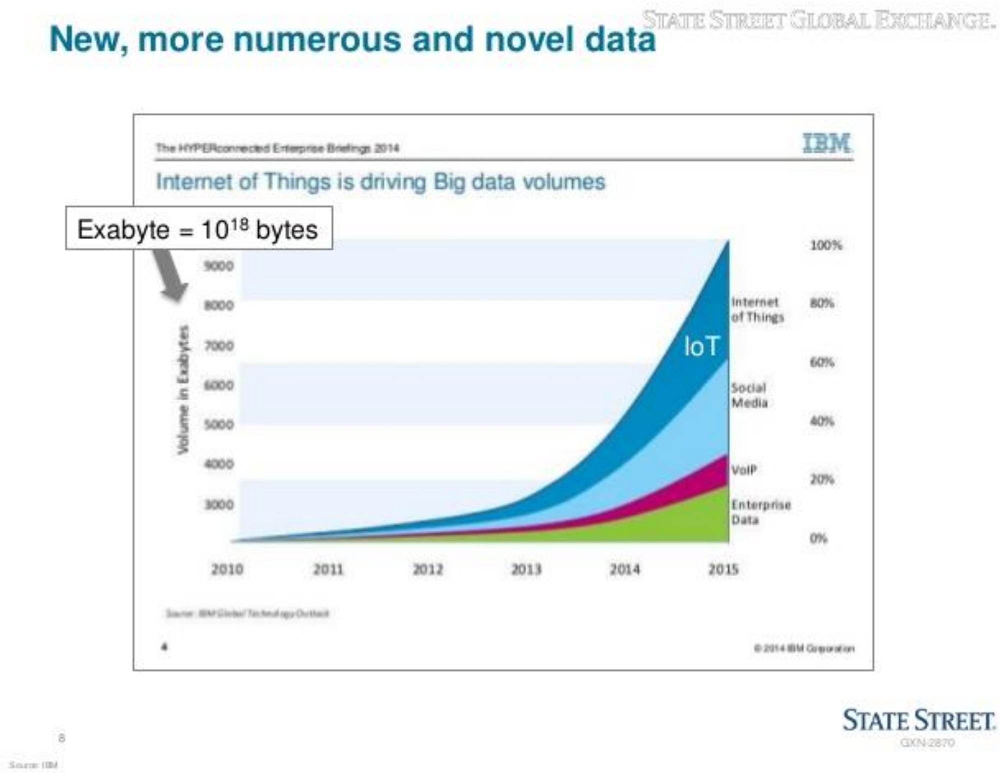
Data is being created every minute of every day without us even noticing it. Given how much information is floating around these days, it's tempting to talk about big data only in terms of size. Big data describes the massive avalanche of digital activity pulsating through cables and airwaves, but it also describes all the things we were never able to measure before. With every status we share, every article we read or every photo we upload, we are creating a digital trail that tells a story. Below, we explore how much data is generated in one minute.

EVERY MINUTE OF THE DAY
PINTEREST USERS PIN 3,472 images
YOUTUBE USERS UPLOAD 72 HRS. OF NEW VIDEO.
EMAIL USERS SEND 204,000,000 MESSAGES.
Google RECEIVES OVER 4,000,000 SEARCH QUERIES.
FACEBOOK USERS SHARE 2,460,000 PIECES OF CONTENT.
VINE USERS SHARE 8,333 VIDEOS.
SKYPE USERS CONNECT FOR 23,300 HOURS.
TINDER USERS SWIPE 416,667 TIMES.
YELP USERS POST 26,380 REVIEWS.
WHATSAPP USERS SHARE 347,222 PHOTOS.
APPLE USERS DOWNLOAD 48,000 apps.
PANDORA USERS LISTEN TO 61,141 HOURS OF music.
AMAZON MAKES $83,000 IN ONLINE SALES.
INSTAGRAM USERS» POST 216,000 NEW PHOTOS.
TWITTER USERS TWEET 277,000 TIMES.

THE GLOBAL INTERNET POPULATION GREW 14.3% FROM 2011 - 2013 AND NOW REPRESENTS 2.4 BILLION PEOPLE.

With each click, share and like, the world's data pool is expanding faster than we can comprehend. Businesses today are paying attention to scores of data sources to make crucial decisions about the future. The team at Domo can help your business make sense of this endless stream of data by providing executives with all their critical information in one intuitive platform. Domo delivers the insights you need to transform the way you run your business. Learn more at www.domo.com.

SOURCES:
BITS.BLOGS.NYTIMES.COM, INTEL.COM, APPLE.COM, TIME.COM, DAILYMAIL.CO.UK, SKYPE.COM, STATISTICBRAIN.COM



STATE STREET GLOBAL EXCHANGE.

New, more numerous and novel data

The HYPERconnected Enterprise Briefings 2014

IBM

Internet of Things is driving Big data volumes

Exabyte = 10^{18} bytes

Source: IBM Global Technology Outlook

4

© 2014 IBM Corporation

8

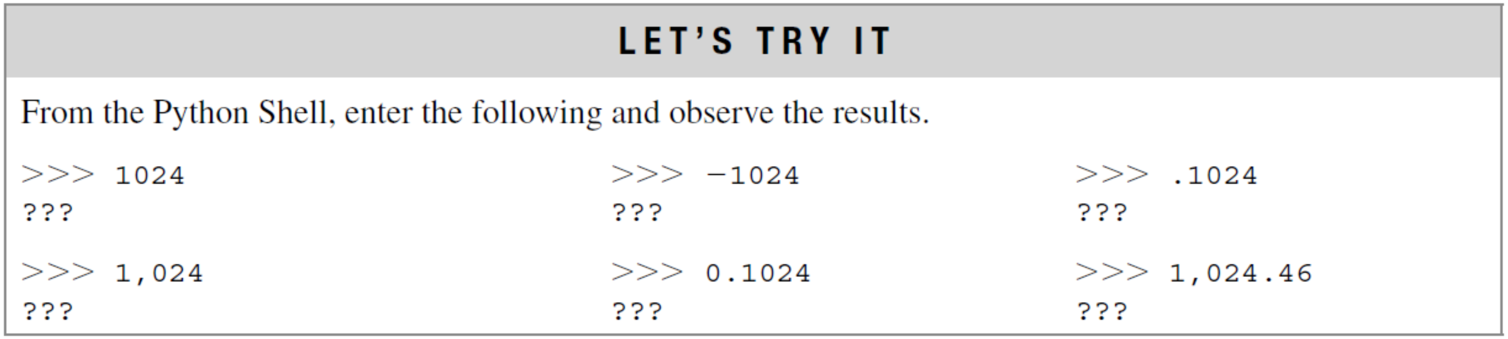
Source: IBM

STATE STREET.
GXN-2870

¿How to we manipulate data in a language?

Fundamental Concepts

- A **literal** is a sequence of one or more characters that stands for itself.
- A **numeric literal** is a literal containing only the digits 0–9, a sign character (1 or 2) and a possible decimal point. Commas are never used in numeric literals.

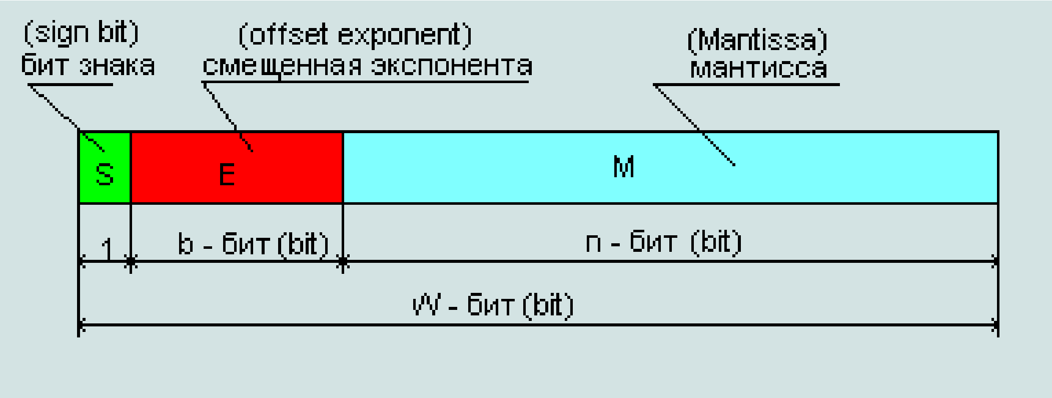


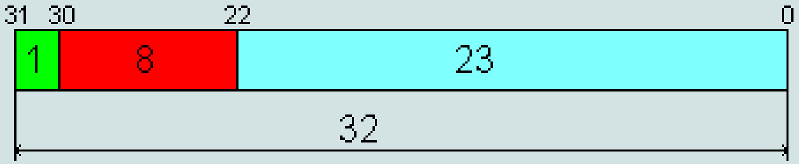
LET'S TRY IT

From the Python Shell, enter the following and observe the results.

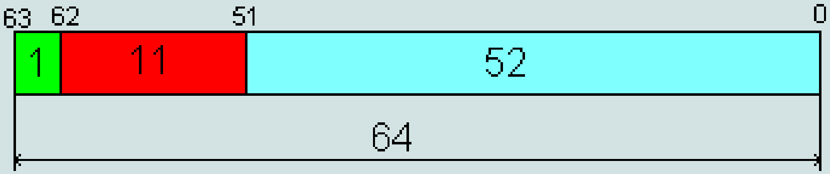
```
>>> 1024                        >>> −1024                  >>> .1024
???                             ???                        ???

>>> 1,024                       >>> 0.1024                 >>> 1,024.46
???                             ???                        ???
```

IEEE 754



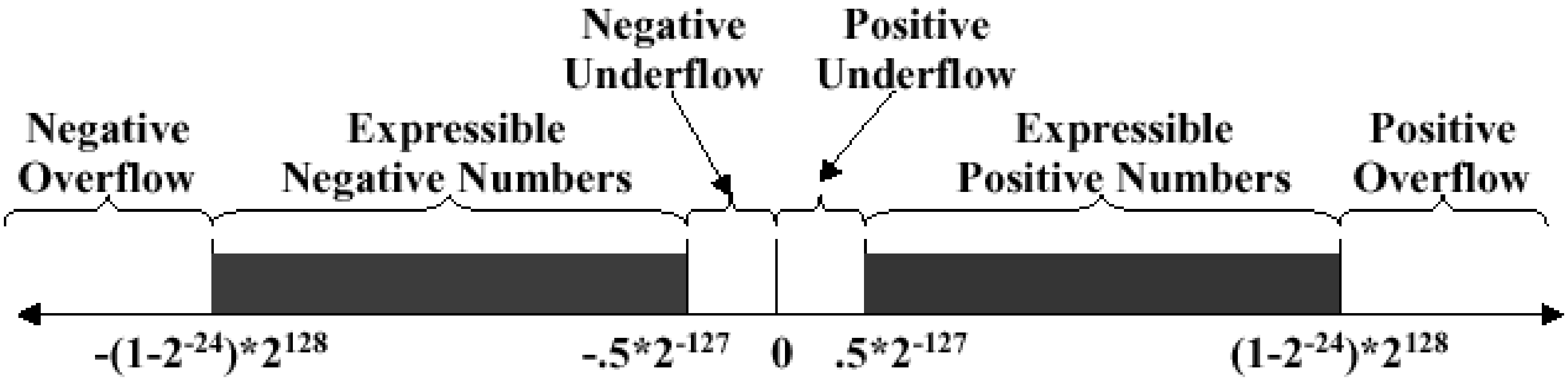


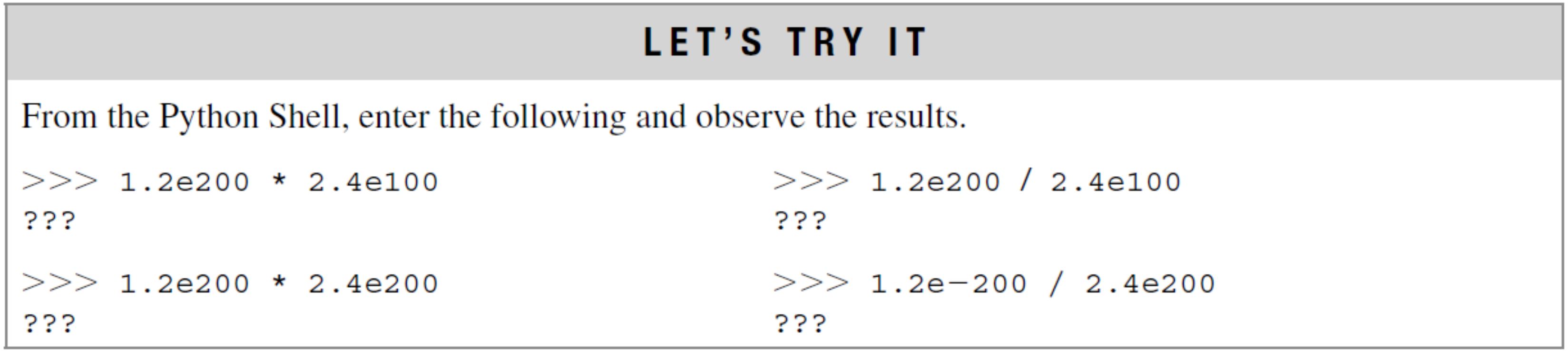
$$F = (-1)^S 2^{(E-127)} (1+M/2^{23})$$



$$F = (-1)^S \ 2^{(E-1023)} (1+M/2^{52})$$

Overflow and underflow



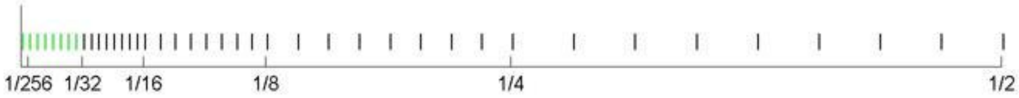


LET'S TRY IT

From the Python Shell, enter the following and observe the results.

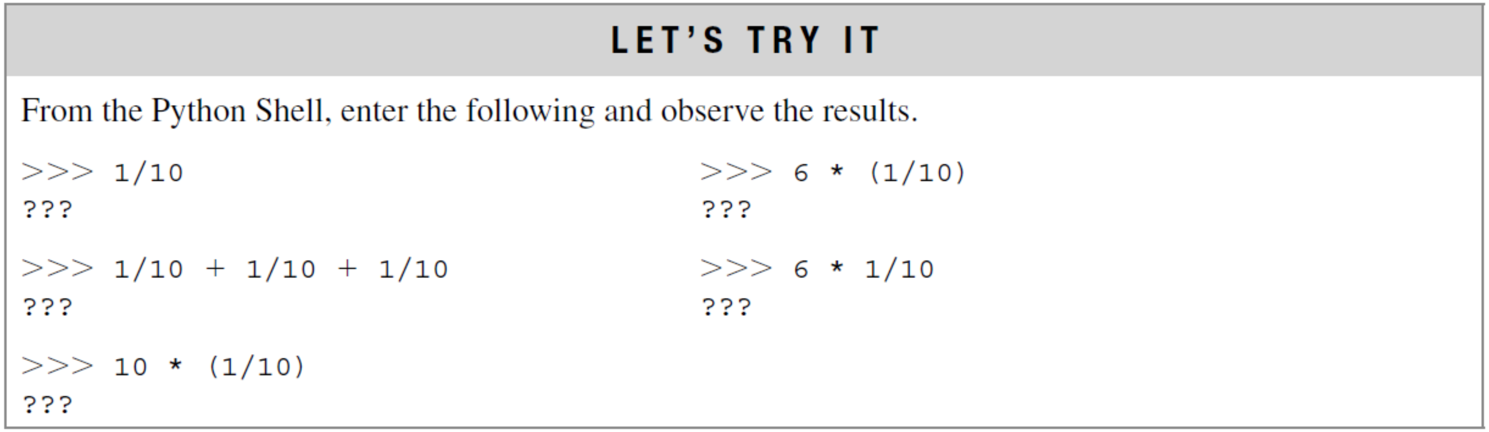
```
>>> 1.2e200 * 2.4e100            >>> 1.2e200 / 2.4e100
???                              ???

>>> 1.2e200 * 2.4e200            >>> 1.2e−200 / 2.4e200
???                              ???
```

Precision and rounding



```
>>> 1/3 + 1/3 + 1/3 + 1/3 + 1/3 + 1/3
1.9999999999999998
```

```
>>> 6 * (1/3)
2.0
```



LET'S TRY IT

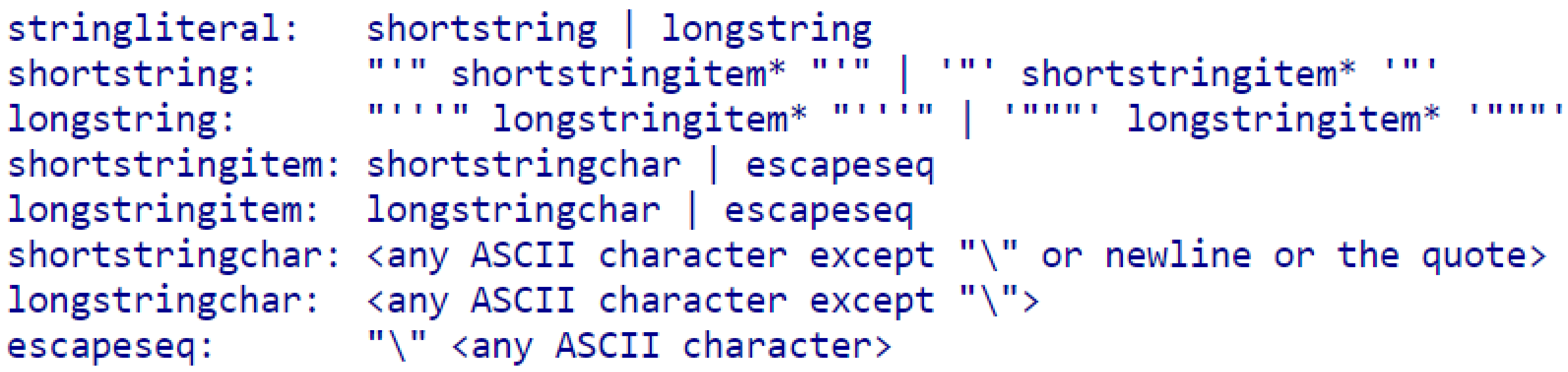
From the Python Shell, enter the following and observe the results.

```
>>> 1/10                              >>> 6 * (1/10)
???                                   ???

>>> 1/10 + 1/10 + 1/10                >>> 6 * 1/10
???                                   ???

>>> 10 * (1/10)
???
```

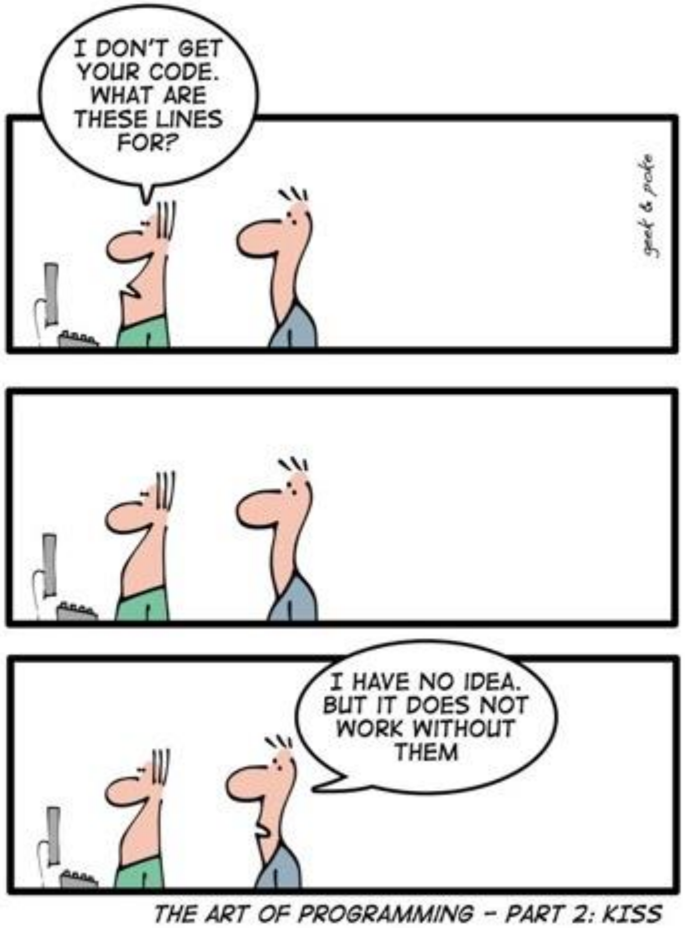
Literal Strings

- A **string literal**, or **string**, is a sequence of characters denoted by a pair of matching single or double (and sometimes triple) quotes in Python.

```
'Hello' 'Smith, John' "Baltimore, Maryland 21210"

"Jennifer Smith's Friend"          ' '
                                   ''
```



```
stringliteral:   shortstring | longstring
shortstring:     "'" shortstringitem* "'" | '"' shortstringitem* '"'
longstring:      "'''" longstringitem* "'''" | '"""' longstringitem* '"""'
shortstringitem: shortstringchar | escapeseq
longstringitem:  longstringchar | escapeseq
shortstringchar: <any ASCII character except "\" or newline or the quote>
longstringchar:  <any ASCII character except "\">
escapeseq:       "\" <any ASCII character>
```



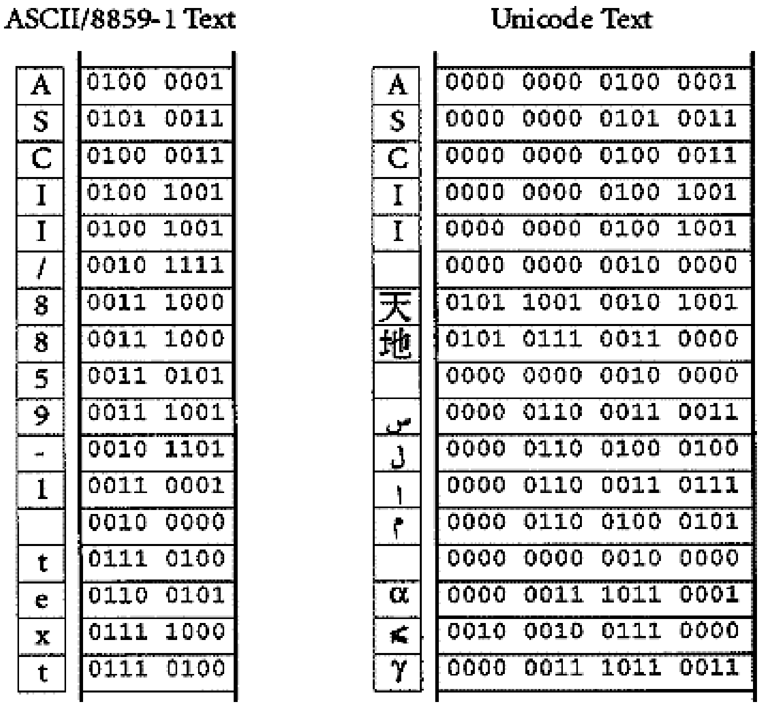
I DON'T GET YOUR CODE. WHAT ARE THESE LINES FOR?
I HAVE NO IDEA. BUT IT DOES NOT WORK WITHOUT THEM
geek & poke
THE ART OF PROGRAMMING – PART 2: KISS

LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> print('Hello')
???

>>> print("Hello")
???

>>> print('Hello")
???

>>> print("Let's Go!')
???

>>> print('Let's Go')
???

>>> print("Let's go!")
???
```

Encoding Characters

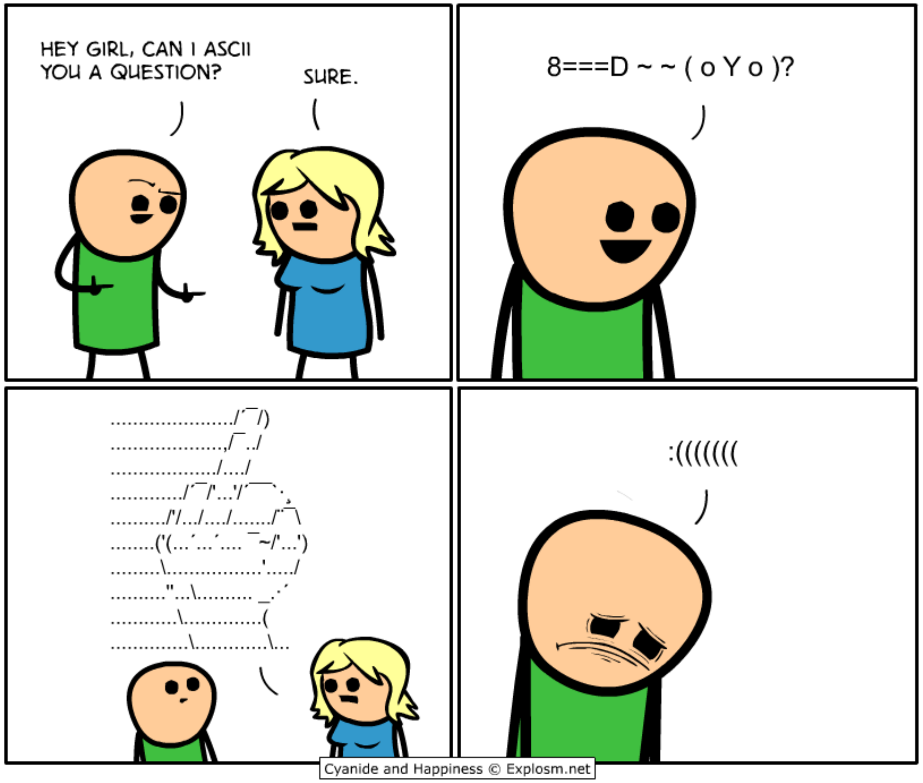


ASCII/8859-1 Text

A	0100 0001
S	0101 0011
C	0100 0011
I	0100 1001
I	0100 1001
/	0010 1111
8	0011 1000
8	0011 1000
5	0011 0101
9	0011 1001
-	0010 1101
1	0011 0001
	0010 0000
t	0111 0100
e	0110 0101
x	0111 1000
t	0111 0100

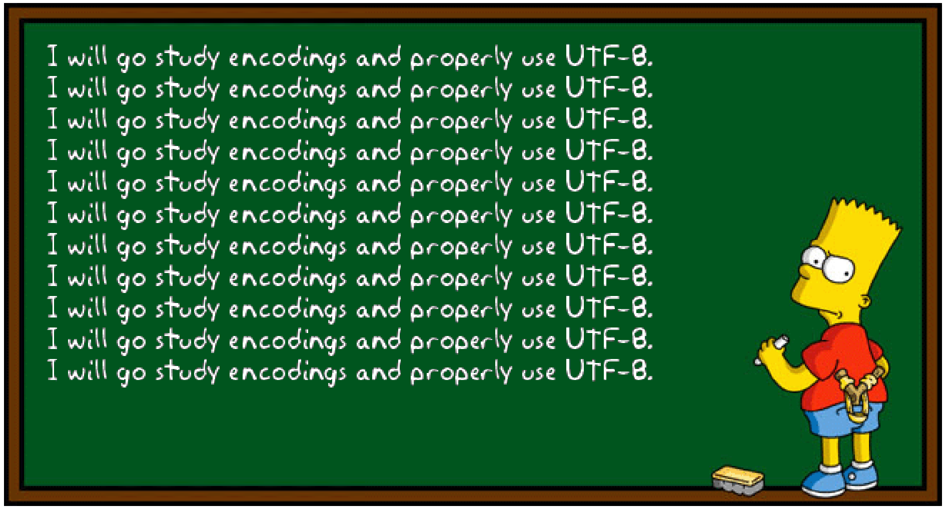
Unicode Text

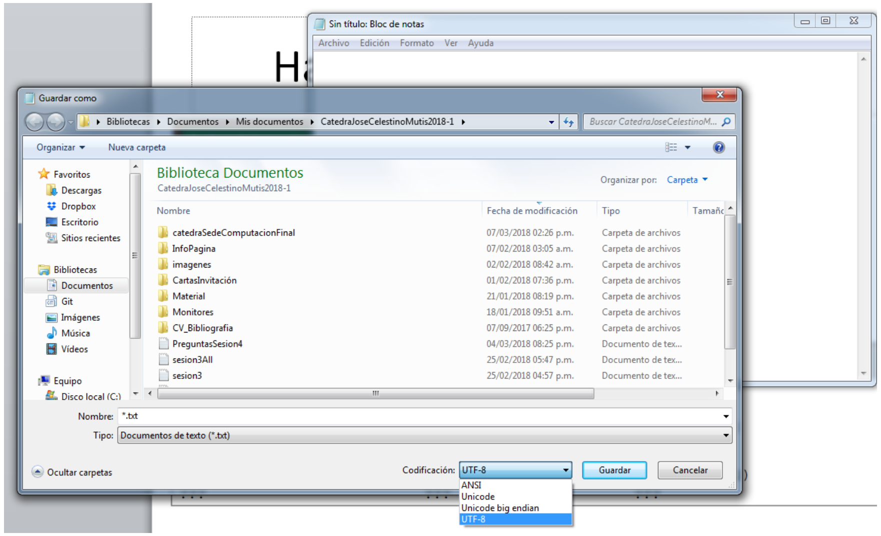
A	0000 0000 0100 0001
S	0000 0000 0101 0011
C	0000 0000 0100 0011
I	0000 0000 0100 1001
I	0000 0000 0100 1001
	0000 0000 0010 0000
天	0101 1001 0010 1001
地	0101 0111 0011 0000
	0000 0000 0010 0000
س	0000 0110 0011 0011
ل	0000 0110 0100 0100
ا	0000 0110 0010 0111
م	0000 0110 0100 0101
	0000 0000 0010 0000
α	0000 0011 1011 0001
≰	0010 0010 0111 0000
γ	0000 0011 1011 0011


HEY GIRL, CAN I ASCII YOU A QUESTION?
SURE.
8===D ~ ~ (o Y o)?
:(((((((
Cyanide and Happiness © Explosm.net

http://www.chris.com/ascii/

https://unicode-table.com/en/#katakana


I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.



Nombre	Fecha de modificación	Tipo	Tamaño
catedraSedeComputacionFinal	07/03/2018 02:26 p.m.	Carpeta de archivos	
InfoPagina	07/02/2018 03:05 a.m.	Carpeta de archivos	
imagenes	02/02/2018 08:42 a.m.	Carpeta de archivos	
CartasInvitación	01/02/2018 07:36 p.m.	Carpeta de archivos	
Material	21/01/2018 08:19 p.m.	Carpeta de archivos	
Monitores	18/01/2018 09:51 a.m.	Carpeta de archivos	
CV_Bibliografia	07/09/2017 06:25 p.m.	Carpeta de archivos	
PreguntasSesion4	04/03/2018 08:25 p.m.	Documento de tex...	
sesion3All	25/02/2018 05:47 p.m.	Documento de tex...	
sesion3	25/02/2018 04:57 p.m.	Documento de tex...	

Nombre: *.txt

Tipo: Documentos de texto (*.txt)

Codificación: UTF-8

- ANSI
- Unicode
- Unicode big endian
- UTF-8

LET'S TRY IT

From the Python Shell, enter the following and observe the results.

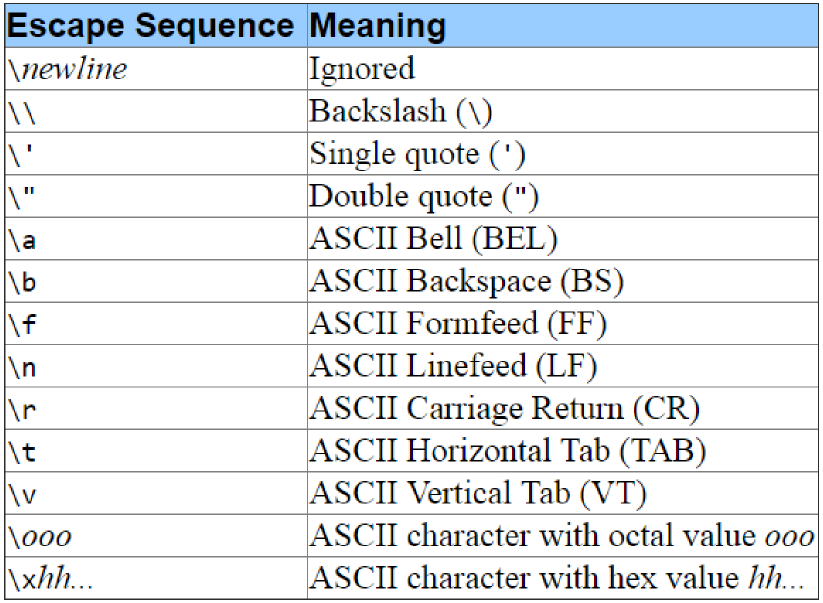
```
>>> ord('1')
???

>>> ord('2')
???
```

```
>>> chr(65)
???

>>> chr(90)
???
```

```
>>> chr(97)
???

>>> chr(122)
???
```

Control Characters

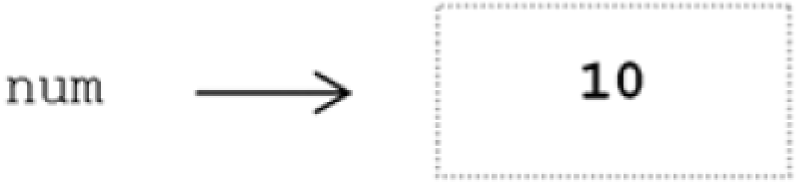
- **Control characters** are nonprinting characters used to *control* the display of output (among other things). An **escape sequence** is a string of one or more characters used to denote control characters.

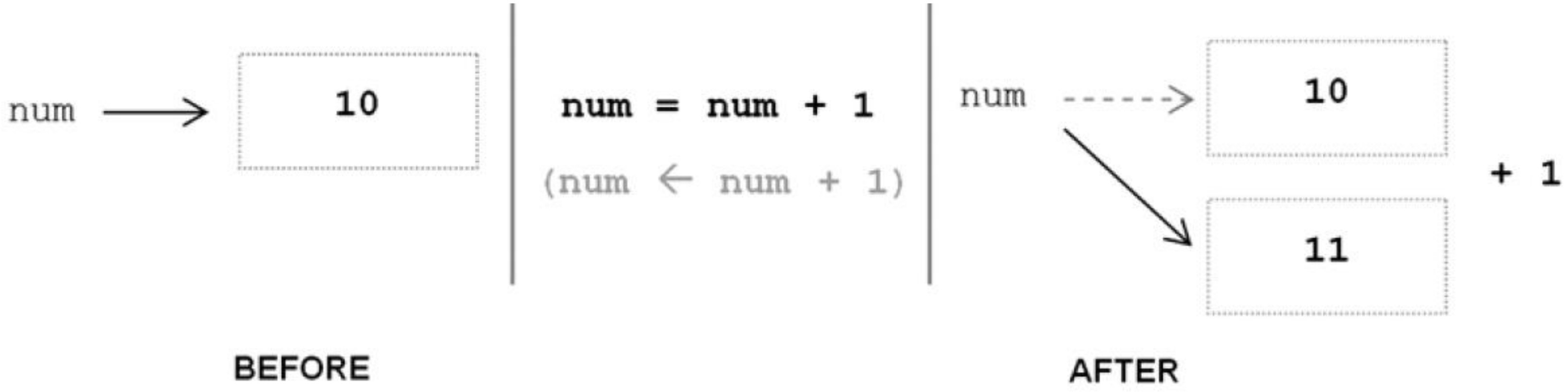


Escape Sequence	Meaning
newline	Ignored
\\\\	Backslash (\\)
\\'	Single quote (')
\\"	Double quote (")
\\a	ASCII Bell (BEL)
\\b	ASCII Backspace (BS)
\\f	ASCII Formfeed (FF)
\\n	ASCII Linefeed (LF)
\\r	ASCII Carriage Return (CR)
\\t	ASCII Horizontal Tab (TAB)
\\v	ASCII Vertical Tab (VT)
ooo	ASCII character with octal value *ooo*
\\x*hh*...	ASCII character with hex value *hh*...

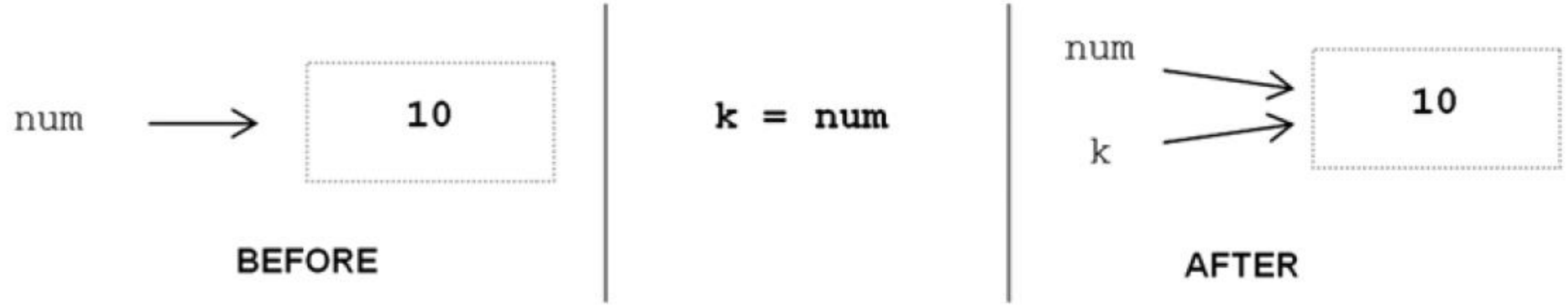
Variables and Identifiers

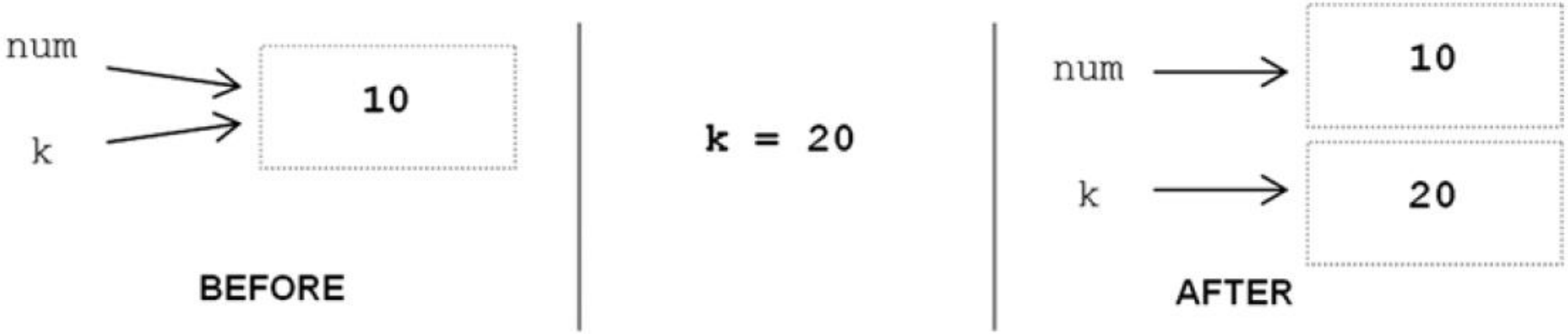
- A **variable** is a name that is associated with a value. The **assignment operator**, =, is used to assign values to variables.



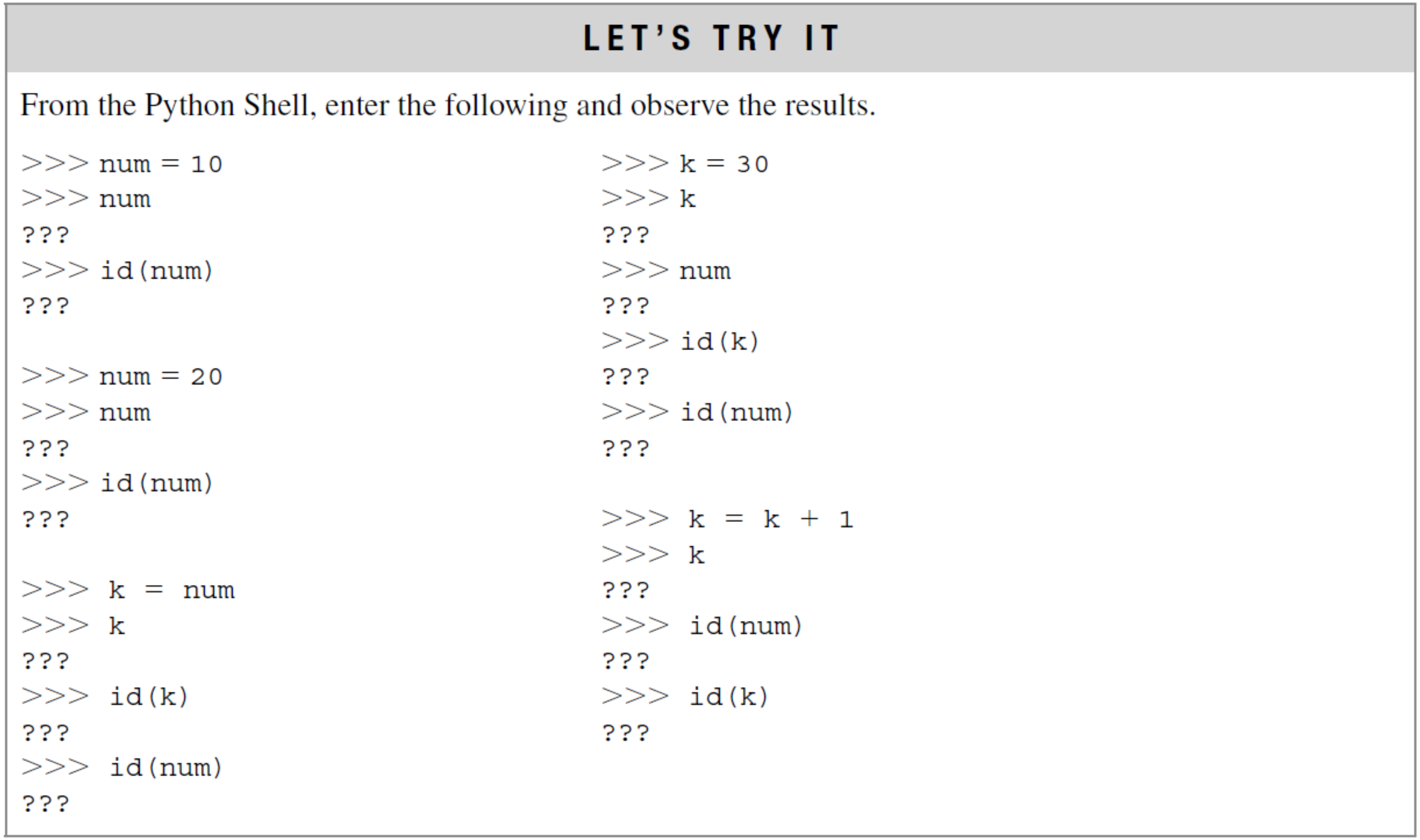


the right side of an assignment is evaluated first, then the result is assigned to the variable on the left





`var = 12`	integer
`var = 12.45`	float
`var = 'Hello'`	string



LET'S TRY IT

From the Python Shell, enter the following and observe the results.

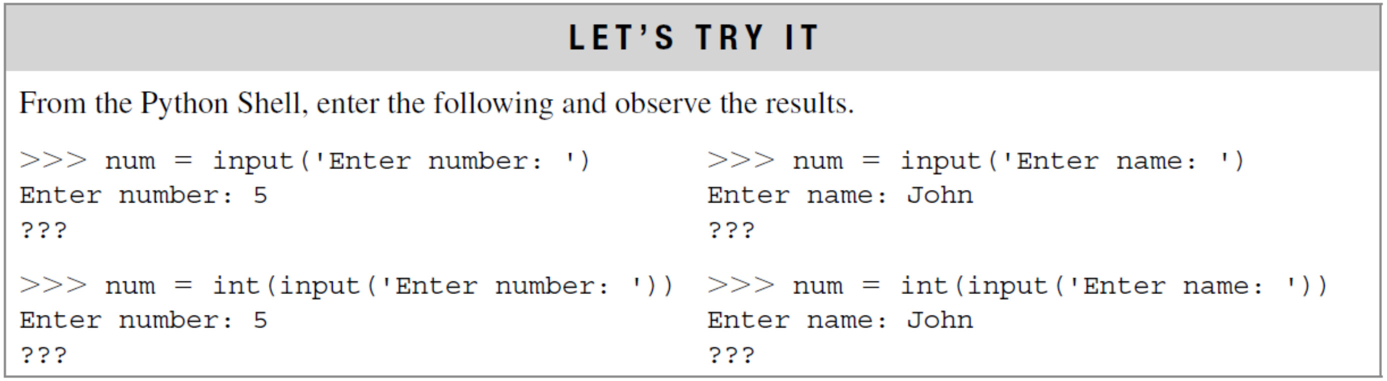
```
>>> num = 10
>>> num
???
>>> id(num)
???

>>> num = 20
>>> num
???
>>> id(num)
???

>>>  k = num
>>>  k
???
>>>  id(k)
???
>>>  id(num)
???
```

```
>>> k = 30
>>> k
???
>>> num
???
>>> id(k)
???
>>> id(num)
???

>>>  k = k + 1
>>>  k
???
>>>  id(num)
???
>>>  id(k)
???
```

id function produces a unique number identifying a specific value (object) in memory.

Keyboard assignation

- All input is returned by the input function as a string type. Built-in functions int() and float() can be used to convert a string to a numeric type.

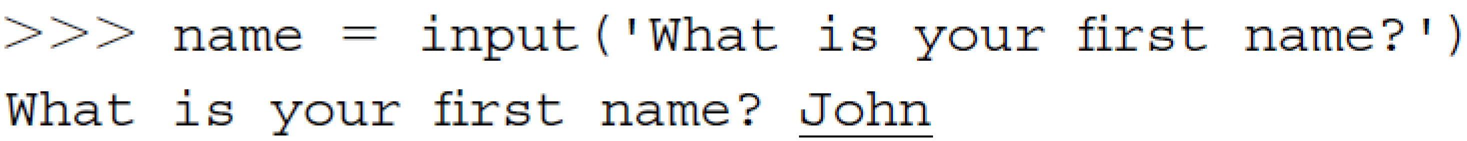


LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> num = input('Enter number: ')
Enter number: 5
???

>>> num = int(input('Enter number: '))
Enter number: 5
???
```

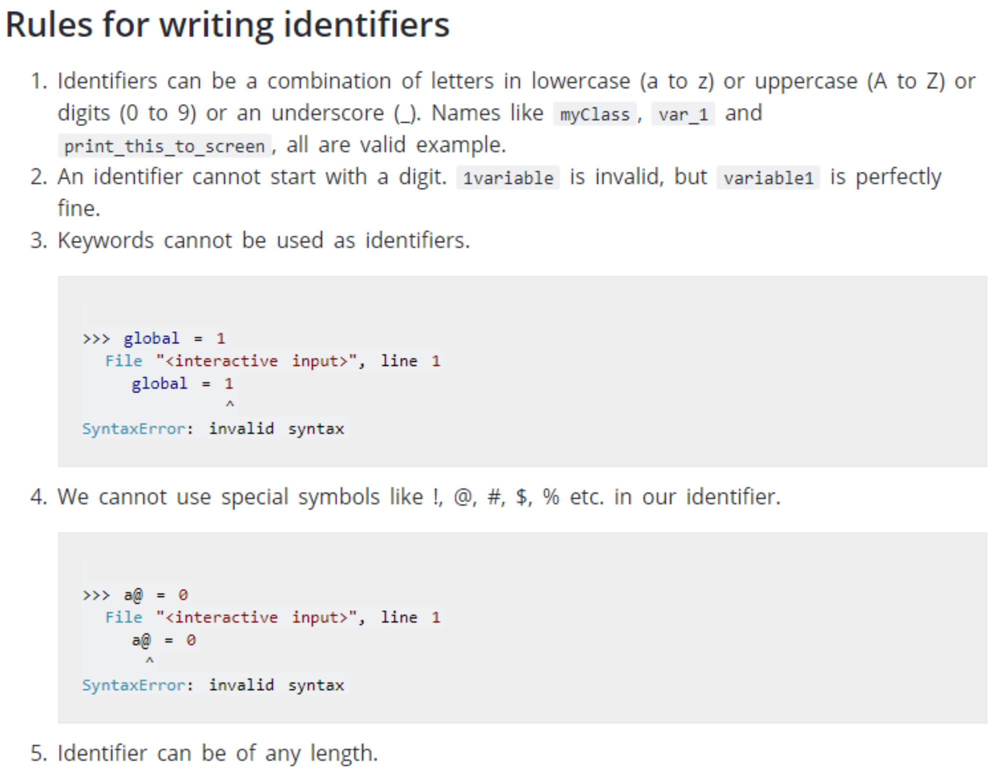
```
>>> num = input('Enter name: ')
Enter name: John
???

>>> num = int(input('Enter name: '))
Enter name: John
???
```



```
>>> name = input('What is your first name?')
What is your first name? John
```

Identifier

- An **identifier** is a sequence of one or more characters used to name a given program element.



Rules for writing identifiers

1. Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). Names like `myClass`, `var_1` and `print_this_to_screen`, all are valid example.
2. An identifier cannot start with a digit. `1variable` is invalid, but `variable1` is perfectly fine.
3. Keywords cannot be used as identifiers.

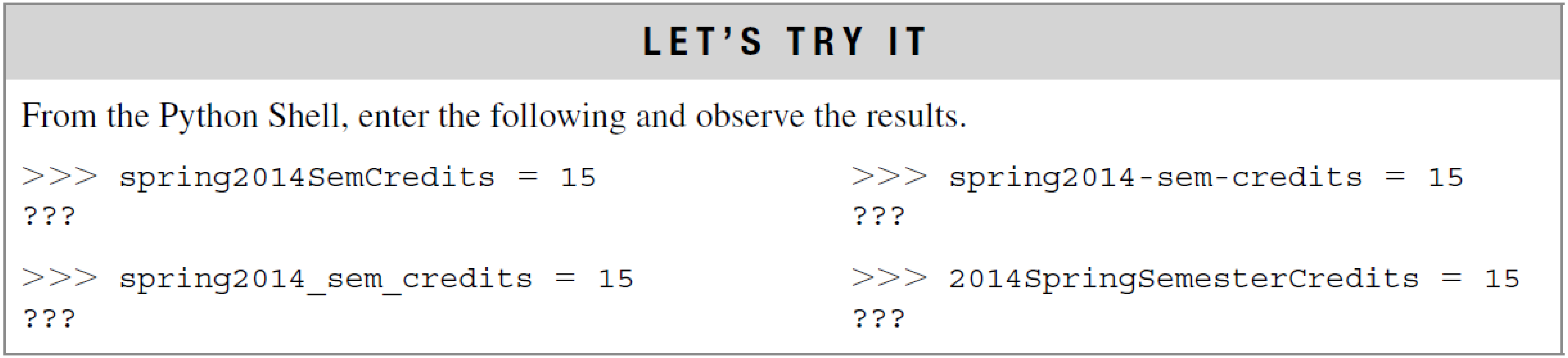
```
>>> global = 1
  File "<interactive input>", line 1
     global = 1
              ^
SyntaxError: invalid syntax
```

4. We cannot use special symbols like !, @, #, $, % etc. in our identifier.

```
>>> a@ = 0
  File "<interactive input>", line 1
     a@ = 0
      ^
SyntaxError: invalid syntax
```

5. Identifier can be of any length.

Identifier



LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> spring2014SemCredits = 15
???
```

```
>>> spring2014_sem_credits = 15
???
```

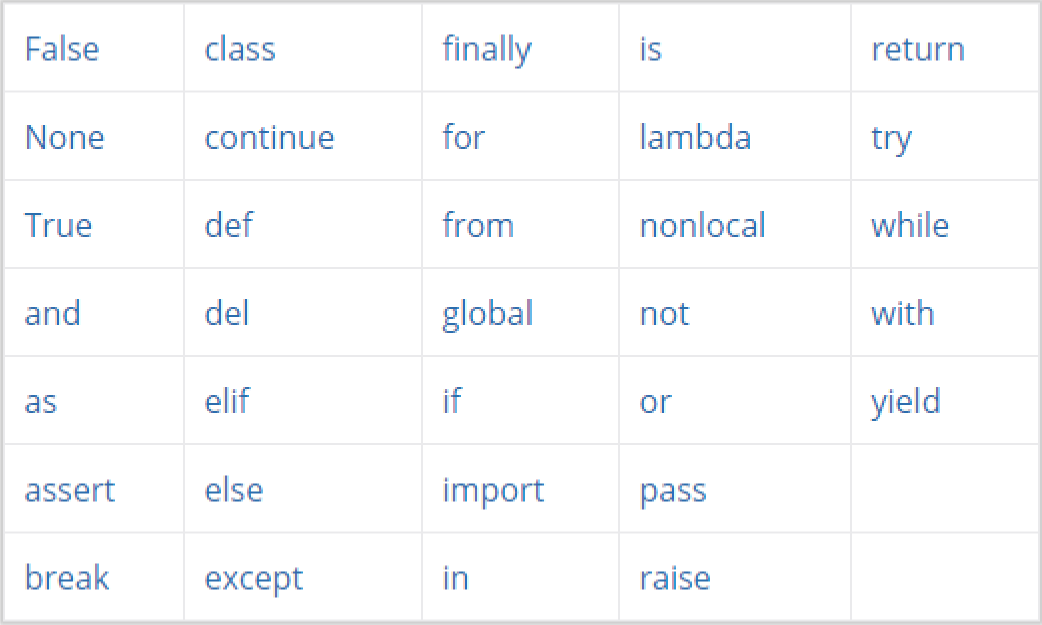
```
>>> spring2014-sem-credits = 15
???
```

```
>>> 2014SpringSemesterCredits = 15
???
```

Keywords

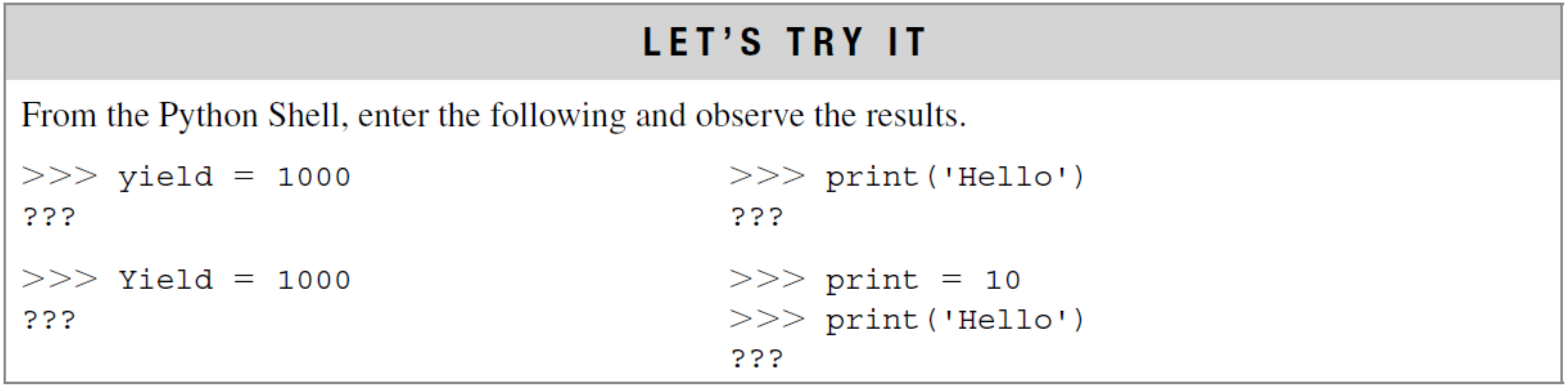
- A **keyword** is an identifier that has predefined meaning in a programming language and therefore cannot be used as a "regular" identifier. Doing so will result in a syntax error.

Keywords in Python programming language



False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Keywords



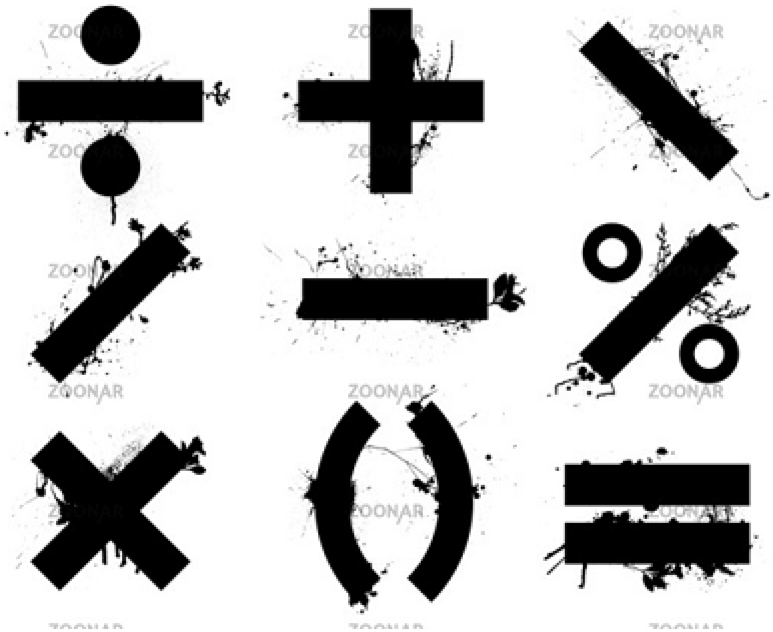
LET'S TRY IT

From the Python Shell, enter the following and observe the results.

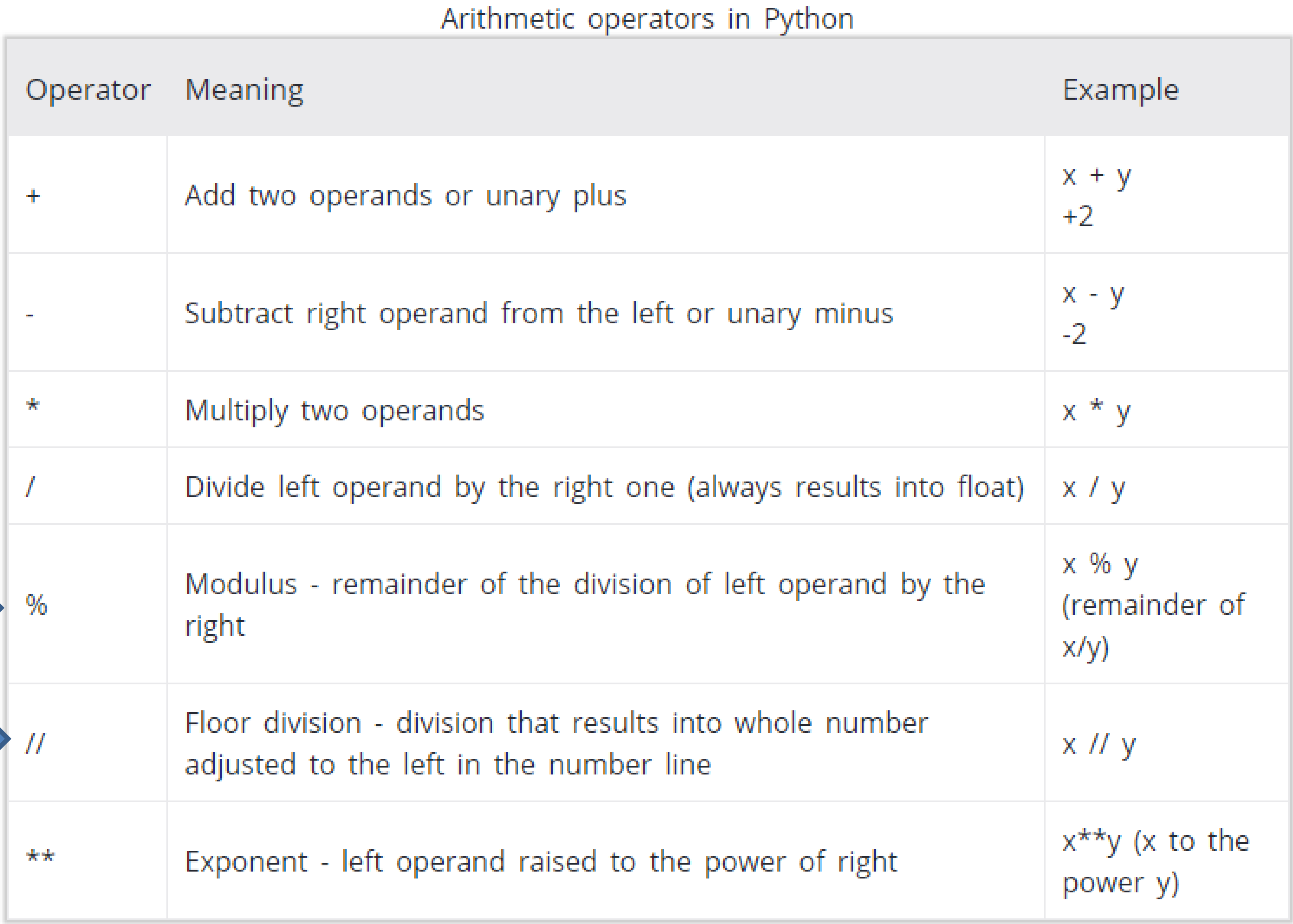
```
>>> yield = 1000                    >>> print('Hello')
???                                 ???

>>> Yield = 1000                    >>> print = 10
???                                 >>> print('Hello')
                                    ???
```

Operators

- An **operator** is a symbol that represents an operation that may be performed on one or more **operands**.
- Operators that take one operand are called **unary operators**.
- Operators that take two operands are called **binary operators**.



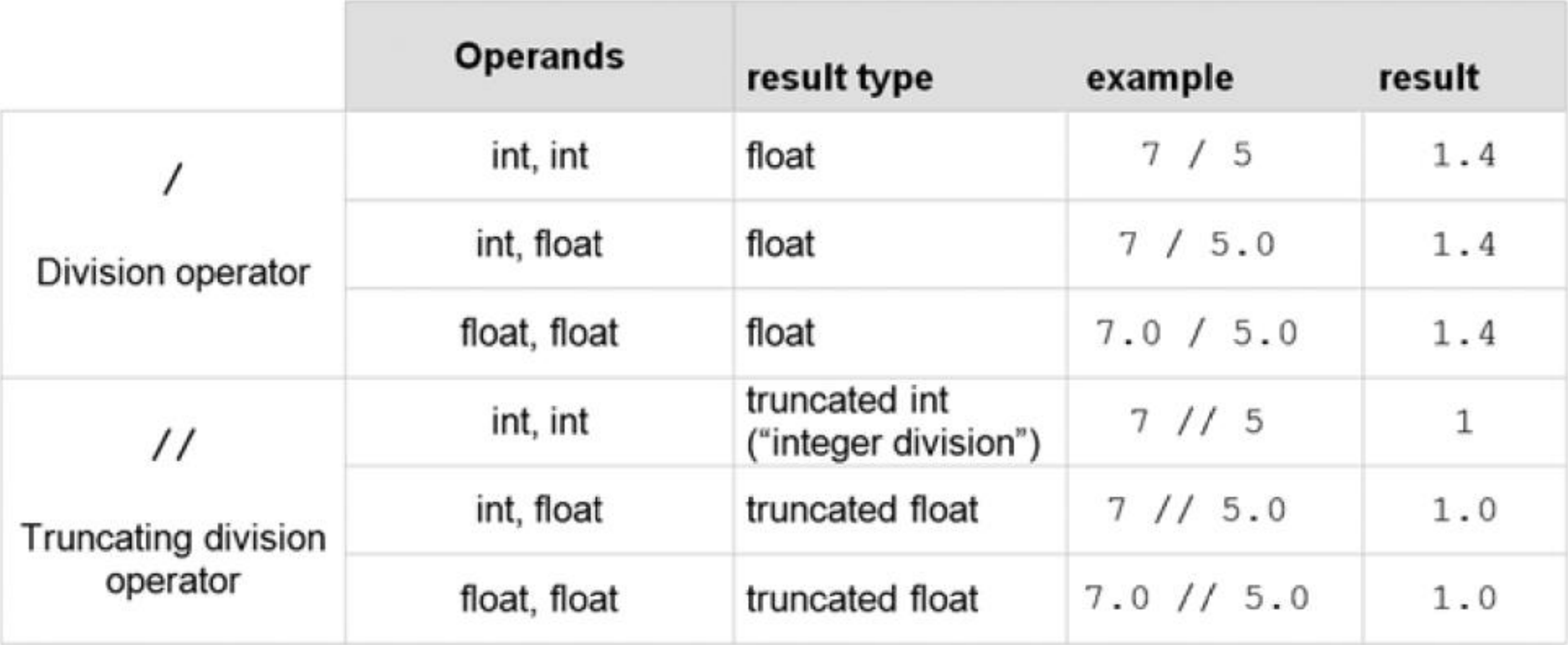
Aritmetic Operators



Arithmetic operators in Python

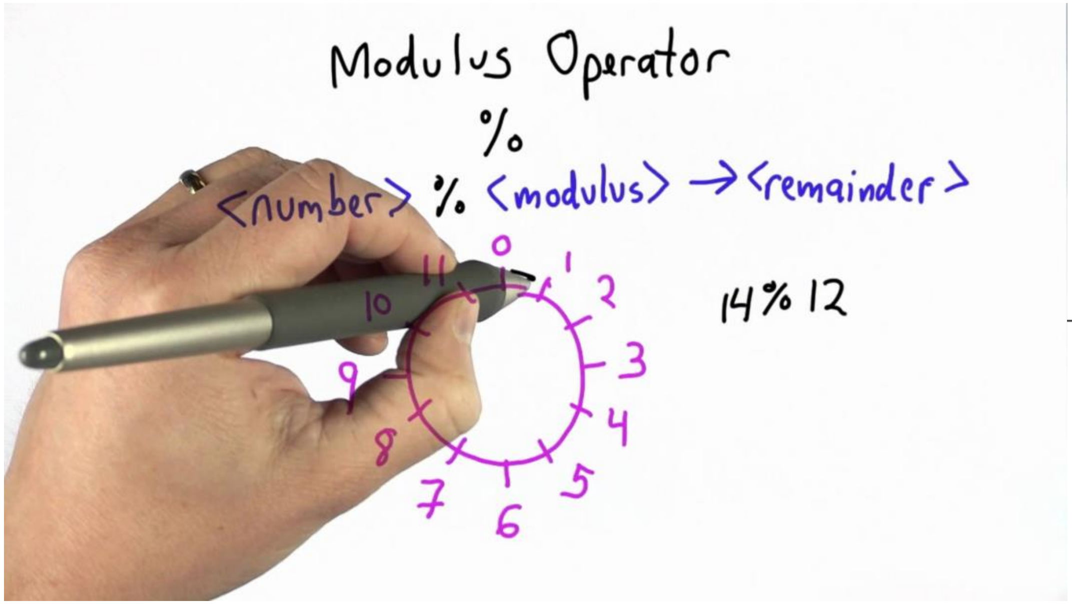
Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

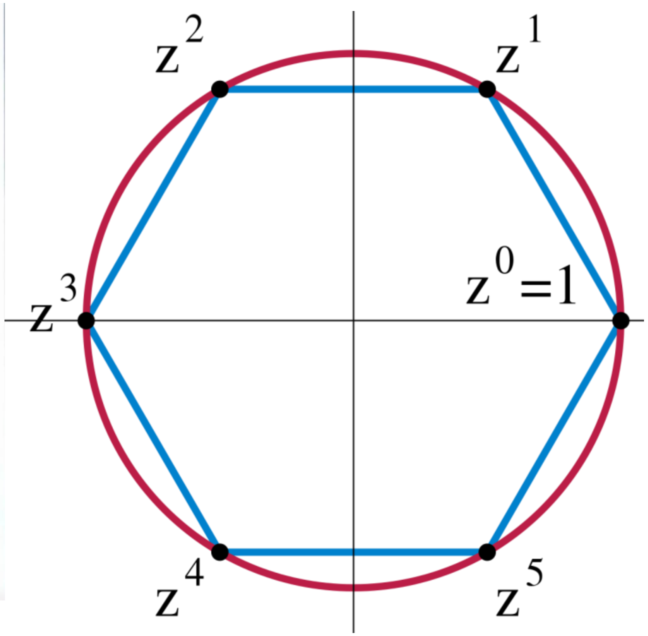
Floor division

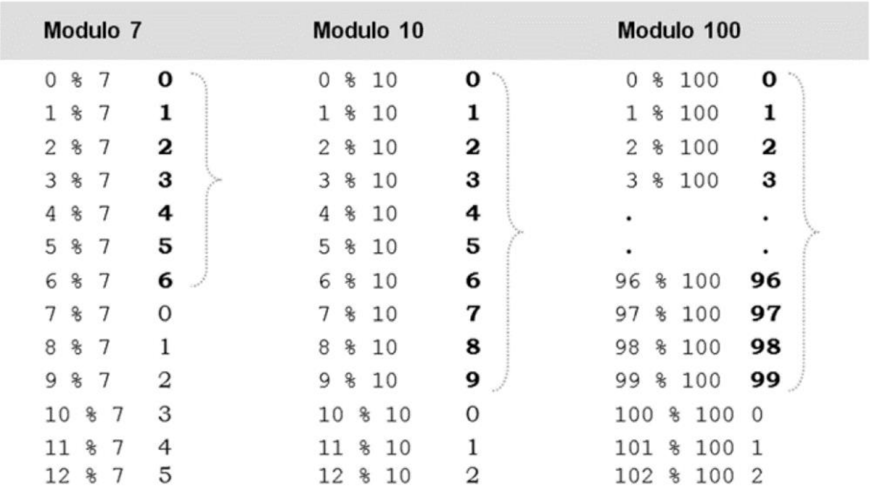


	Operands	result type	example	result
/ Division operator	int, int	float	`7 / 5`	`1.4`
	int, float	float	`7 / 5.0`	`1.4`
	float, float	float	`7.0 / 5.0`	`1.4`
// Truncating division operator	int, int	truncated int (“integer division”)	`7 // 5`	`1`
	int, float	truncated float	`7 // 5.0`	`1.0`
	float, float	truncated float	`7.0 // 5.0`	`1.0`

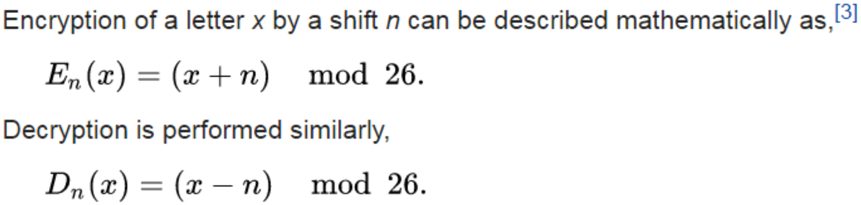
Modulus operator







Modulo 7		Modulo 10		Modulo 100	
0 % 7	**0**	0 % 10	**0**	0 % 100	**0**
1 % 7	**1**	1 % 10	**1**	1 % 100	**1**
2 % 7	**2**	2 % 10	**2**	2 % 100	**2**
3 % 7	**3**	3 % 10	**3**	3 % 100	**3**
4 % 7	**4**	4 % 10	**4**	.	.
5 % 7	**5**	5 % 10	**5**	.	.
6 % 7	**6**	6 % 10	**6**	96 % 100	**96**
7 % 7	0	7 % 10	**7**	97 % 100	**97**
8 % 7	1	8 % 10	**8**	98 % 100	**98**
9 % 7	2	9 % 10	**9**	99 % 100	**99**
10 % 7	3	10 % 10	0	100 % 100	0
11 % 7	4	11 % 10	1	101 % 100	1
12 % 7	5	12 % 10	2	102 % 100	2



Encryption of a letter *x* by a shift *n* can be described mathematically as,[3]

$$E_n(x) = (x + n) \mod 26.$$

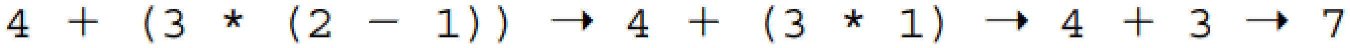
Decryption is performed similarly,

$$D_n(x) = (x - n) \mod 26.$$

Expressions

- An **expression** is a combination of symbols (or single symbol) that evaluates to a value. A **subexpression** is any expression that is part of a larger expression.

4 + (3 * k)

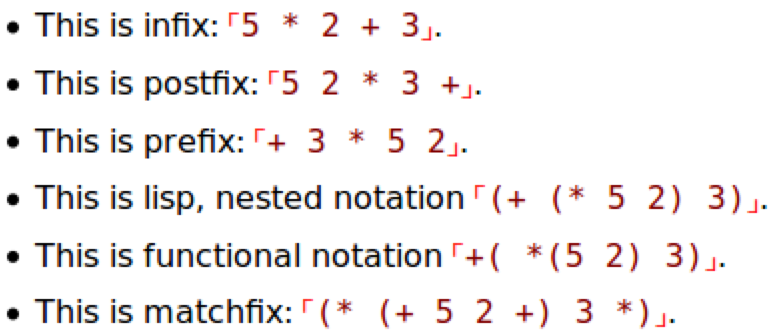


```
4 + (3 * (2 − 1)) → 4 + (3 * 1) → 4 + 3 → 7
```

4 + 3 * 2 -1

Operator precedence

- Expressions in pyhton use infix notation

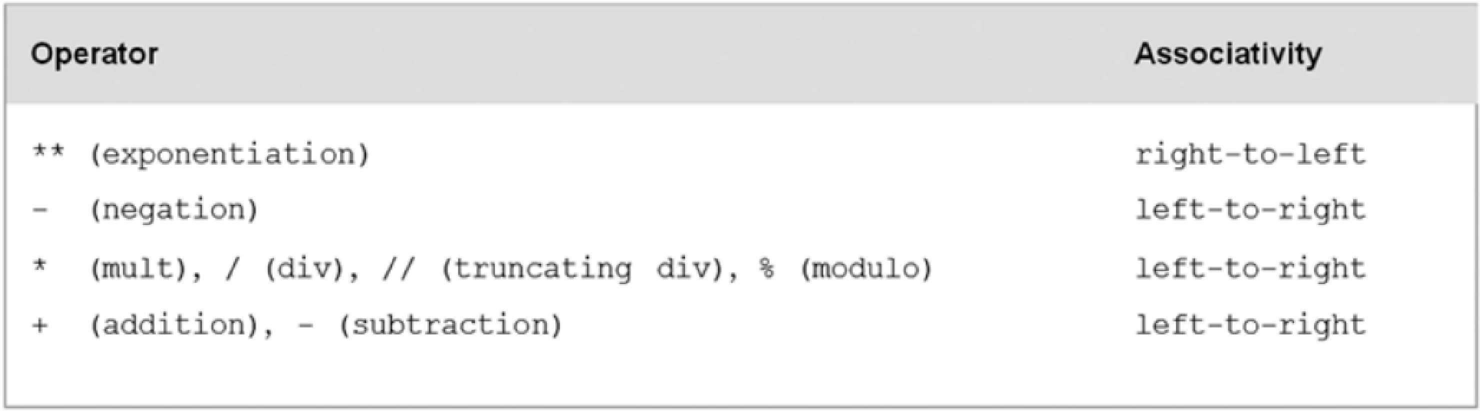


- This is infix: ⌜5 * 2 + 3⌟.
- This is postfix: ⌜5 2 * 3 +⌟.
- This is prefix: ⌜+ 3 * 5 2⌟.
- This is lisp, nested notation ⌜(+ (* 5 2) 3)⌟.
- This is functional notation ⌜+(*(5 2) 3)⌟.
- This is matchfix: ⌜(* (+ 5 2 +) 3 *)⌟.

$$4 + 3 * 5 \qquad \begin{array}{l} 4 + \underbrace{\mathbf{3 * 5}} \rightarrow 4 + 15 \rightarrow 19 \\ \underbrace{\mathbf{4 + 3}} * 5 \rightarrow 7 * 5 \rightarrow 35 \end{array}$$

Operator precedence

- Operator precedence is the relative order that operators are applied in the evaluation of expressions, defined by a given operator precedence table.



Operator	Associativity
`**` (exponentiation)	right-to-left
`-` (negation)	left-to-right
`*` (mult), `/` (div), `//` (truncating div), `%` (modulo)	left-to-right
`+` (addition), `-` (subtraction)	left-to-right

```
4 + 3 * 5 → 4 + 15 → 19

4 + 2 ** 5 // 10 → 4 + 32 // 10 → 4 + 3 → 7
```

Associativity

- What if two operators have the same level of precedence, which one is applied first?
  - If associative law is followed it doesn't matter:

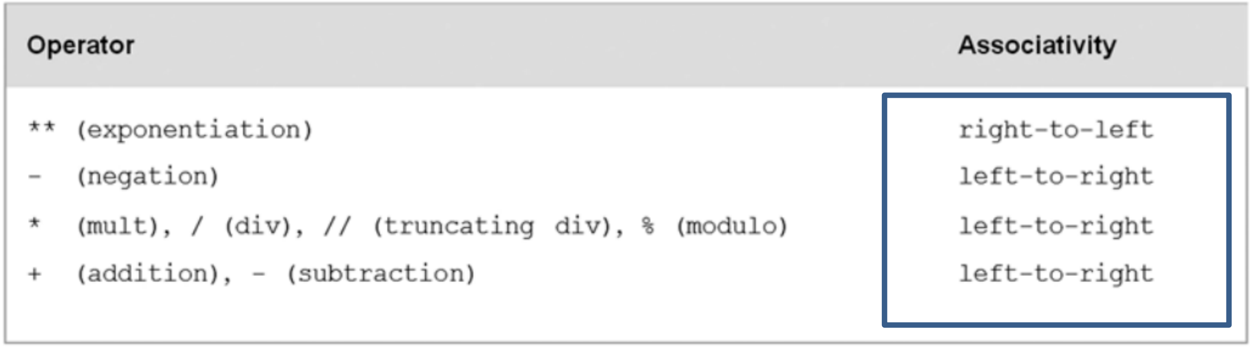
```
(2 + 3) + 4 → 9        2 + (3 + 4) → 9
```

  - But for other operations matter:

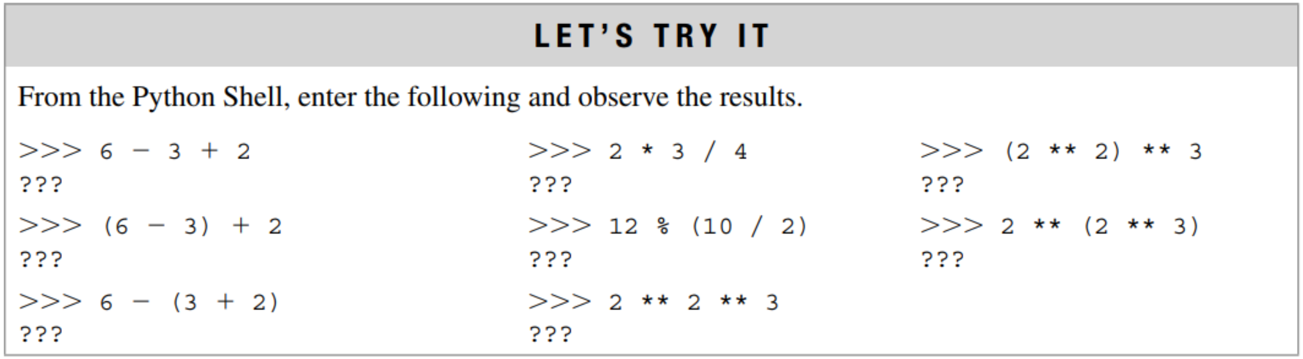
```
(a) (8 − 4) − 2 → 4 − 2 → 2        8 − (4 − 2) → 8 − 2 → 6
(b) (8 / 4) / 2  → 2 / 2 → 1        8 / (4 / 2)  → 8 / 2 → 4
(c) 2 ** (3 ** 2) → 512             (2 ** 3) ** 2 → 64
```

Associativity

- **Operator associativity** is the order that operators are applied when having the same level of precedence, specific to each operator.



Operator	Associativity
`**` (exponentiation)	right-to-left
`-` (negation)	left-to-right
`*` (mult), `/` (div), `//` (truncating div), `%` (modulo)	left-to-right
`+` (addition), `-` (subtraction)	left-to-right

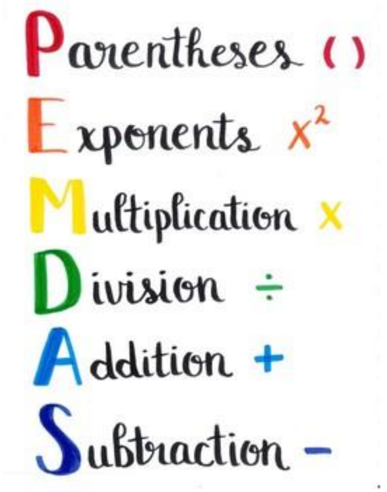


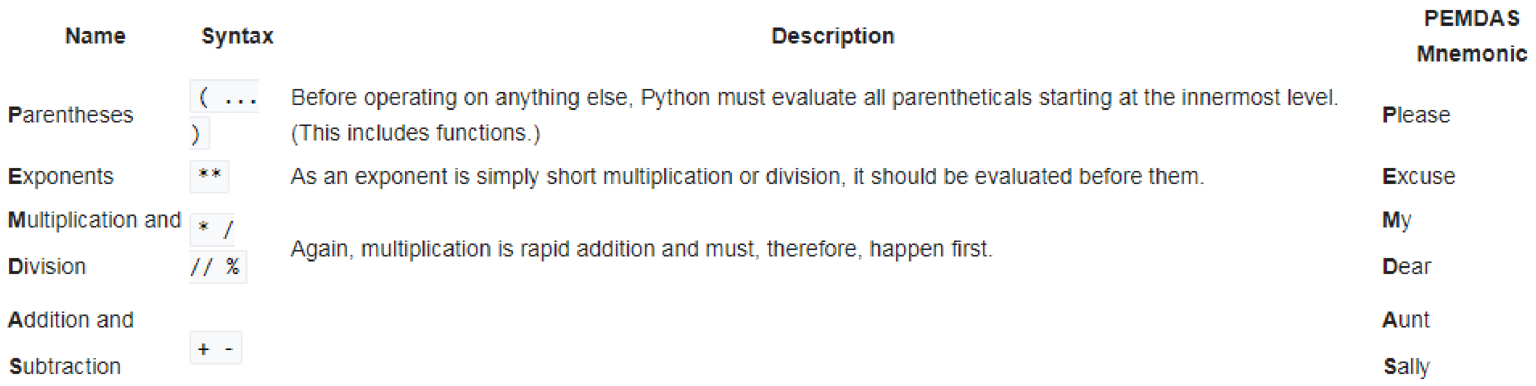
LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> 6 - 3 + 2            >>> 2 * 3 / 4           >>> (2 ** 2) ** 3
???                      ???                     ???
>>> (6 - 3) + 2          >>> 12 % (10 / 2)       >>> 2 ** (2 ** 3)
???                      ???                     ???
>>> 6 - (3 + 2)          >>> 2 ** 2 ** 3
???                      ???
```

PEMDAS





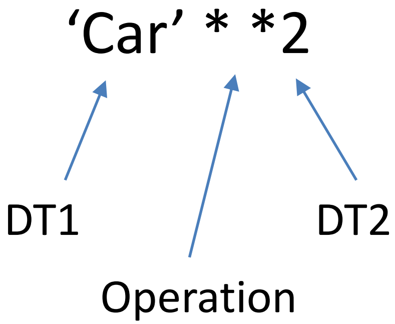
Name	Syntax	Description	PEMDAS Mnemonic
Parentheses	`( ... )`	Before operating on anything else, Python must evaluate all parentheticals starting at the innermost level. (This includes functions.)	Please
Exponents	`**`	As an exponent is simply short multiplication or division, it should be evaluated before them.	Excuse

Multiplication and Division	`* /`
	`// %`
		Again, multiplication is rapid addition and must, therefore, happen first.	My
			Dear

Addition and Subtraction	`+ -`		Aunt Sally

Data type

- A **data type** is a set of *values*, and a set of *operators* that may be applied to those values.

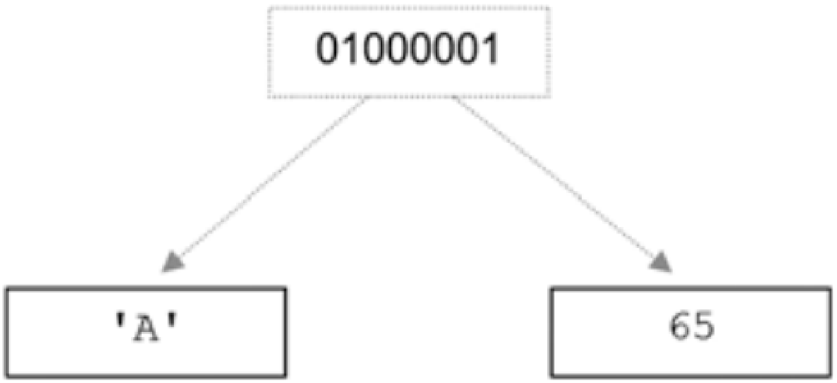


‘Car’ * *2

DT1

DT2

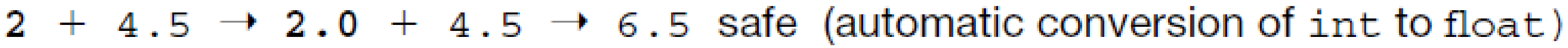
Operation



- Python has **Built-in** types: integer, float and string.
- Python has dynamic typing: data type of a variable depends only on the type of value that the variable is currently holding.

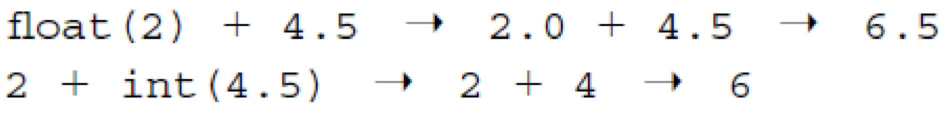
Mixed type expressions

- A **mixed-type** expression is an expression with operands of different type
- CPU only operates in the same type, then conversion should be performed by using:
  - **Coercion:** implicit (automatic) conversion of operands to a common type



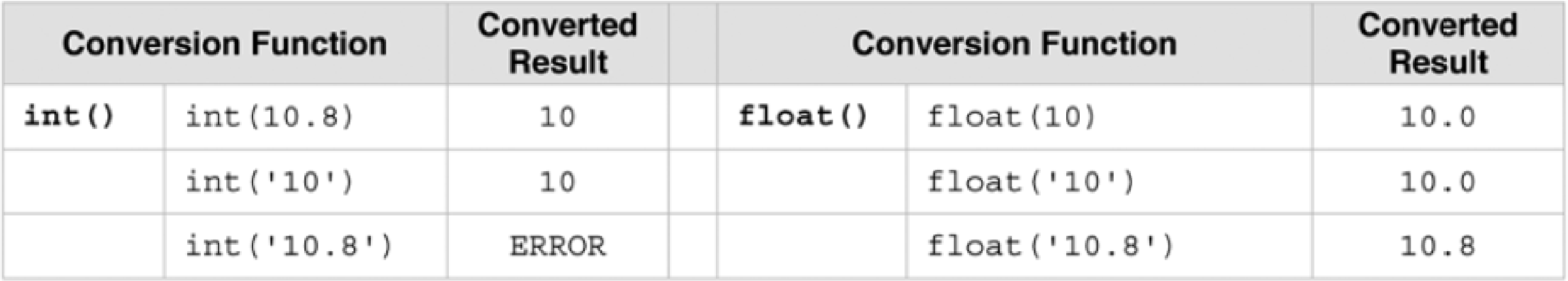
`2 + 4.5 → 2.0 + 4.5 → 6.5` safe (automatic conversion of `int` to `float`)

  - **Type conversion:** Explicit conversion of operands to a specific type



```
float(2) + 4.5  →  2.0 + 4.5  →  6.5
2 + int(4.5)  →  2 + 4  →  6
```

Conversion functions



Conversion Function		Converted Result		Conversion Function		Converted Result
int()	int(10.8)	10		**float()**	float(10)	10.0
	int('10')	10			float('10')	10.0
	int('10.8')	ERROR			float('10.8')	10.8

- Strings can be converted to numeric types.

```
num_credits = int(input('How many credits do you have? '))
```