

# Expressions and Syntax

Carol V. Alexandru-Funakoshi  
Informatics I - 2023

# Expressions and Syntax

$$\sqrt{\frac{1}{2}}$$

$$x + y^2$$

$$\sum_{i=3}^6 i^2$$

$$m c^2$$

$$\sqrt{a^2 + b^2}$$

# Values

True

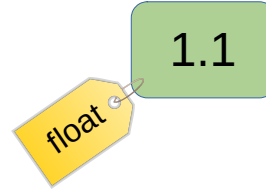
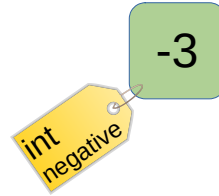
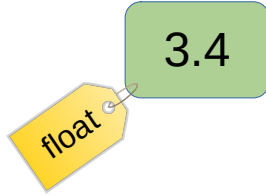
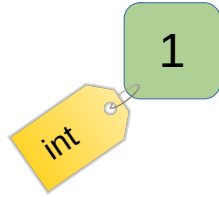
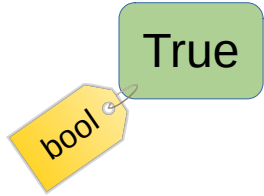
1

3.4

-3

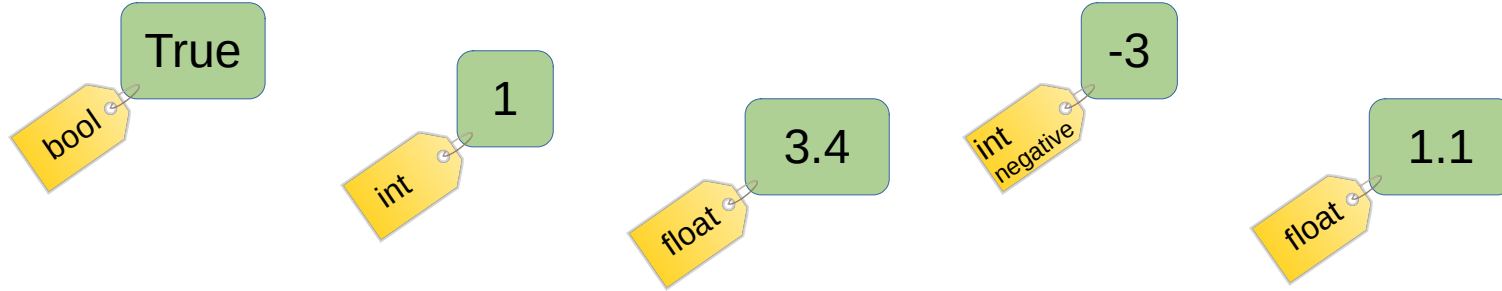
1.1

# Values



# Values

$x + y^2$   
in Python:  
 $x + y^{**2}$



Values “fit in” where they have a compatible type:

Yes:

1 + 3.4 \*\* 2

No:

1 + "Hi"

3.4.lower()

# Understanding an expression

Step 1: Recognize the boxes!

$$\boxed{1} + \boxed{3.4} ** \boxed{2}$$

$$\boxed{\phantom{1}} + \boxed{\phantom{3.4}} ** \boxed{\phantom{2}}$$

# Understanding an expression

Step 1: Recognize the boxes!

$$\boxed{1} + \boxed{3.4} ** \boxed{2}$$

$$\boxed{\phantom{1}} + \boxed{\phantom{3.4}} ** \boxed{\phantom{2}}$$

$$\boxed{\phantom{1}} + \boxed{\phantom{3.4}} ** \boxed{\phantom{2}}$$

# Understanding an expression

Step 1: Recognize the boxes!

$$\boxed{1} + \boxed{3.4} ** \boxed{2}$$

$$\boxed{\phantom{1}} + \boxed{\phantom{3.4}} ** \boxed{\phantom{2}}$$

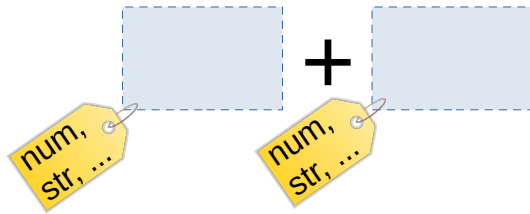
$$\boxed{\phantom{1}} + \boxed{\phantom{3.4}} ** \boxed{\phantom{2}}$$

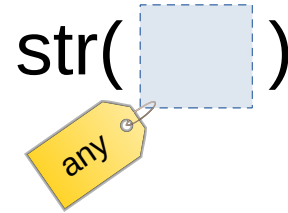
$$\boxed{\phantom{1}} + \boxed{\phantom{3.4}} ** \boxed{\phantom{2}}$$



# Understanding an expression

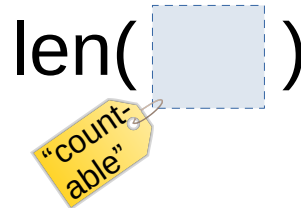
Step 2: Recognize what kinds of values “fit”!


  
num, str, ... + num, str, ...

  
str( any )

  
math.floor( num )

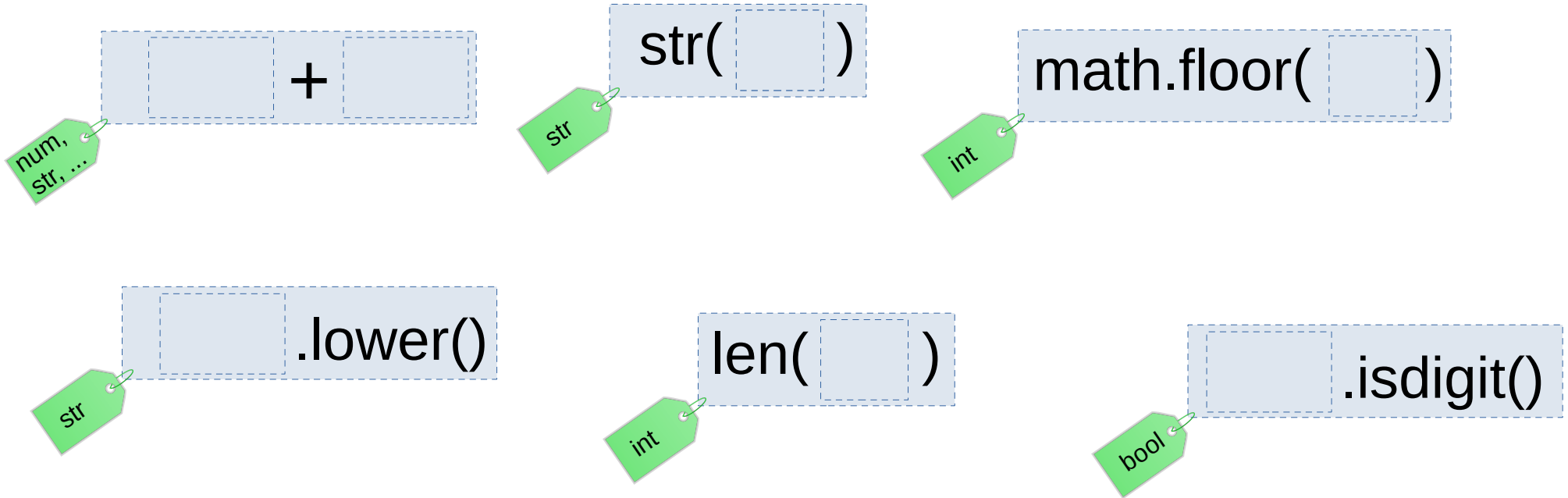
  
str .lower()

  
len( countable )

  
str .isdigit()

# Understanding an expression

Step 3: Recognize what comes out!

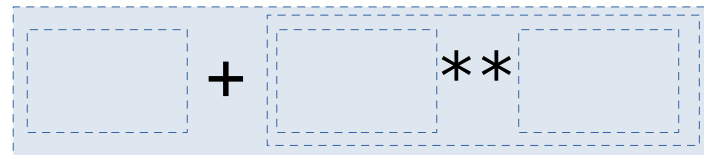


Steps 1, 2 and 3:

$$1 + 3.4 ** 2$$

$$1 + 3.4 ** 2$$

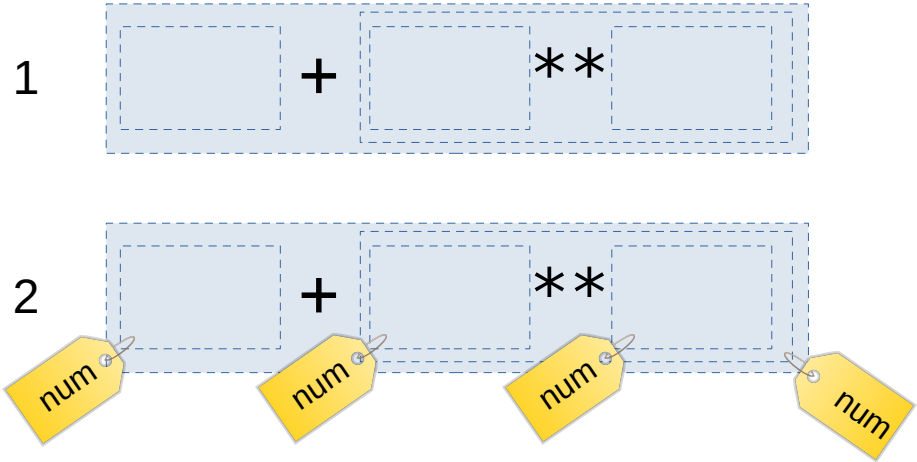
1



Steps 1, 2 and 3:

$$\boxed{1} + \boxed{3.4} ** \boxed{2}$$

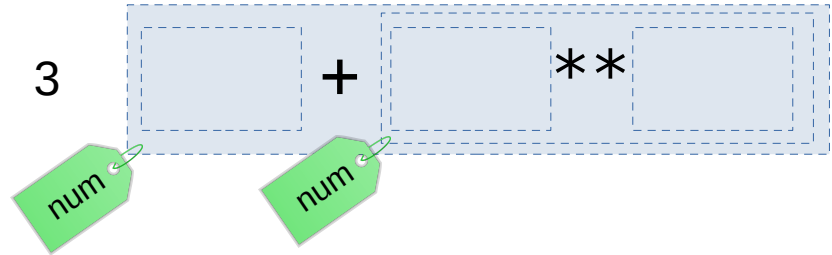
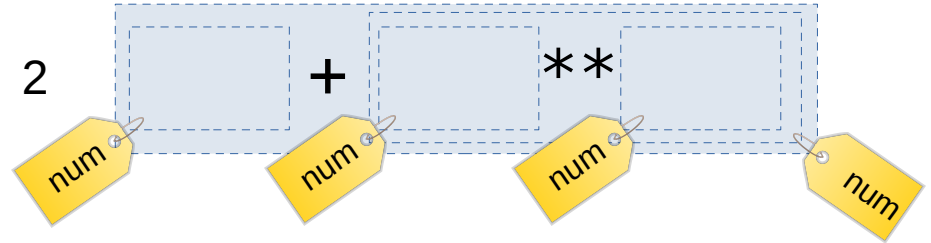
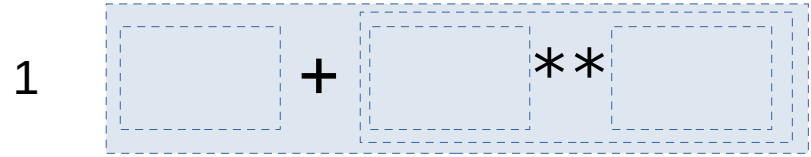
$$1 + 3.4 ** 2$$



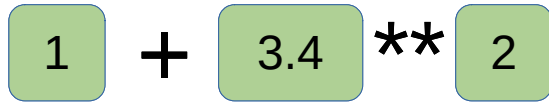
Steps 1, 2 and 3:

$$\boxed{1} + \boxed{3.4} ** \boxed{2}$$

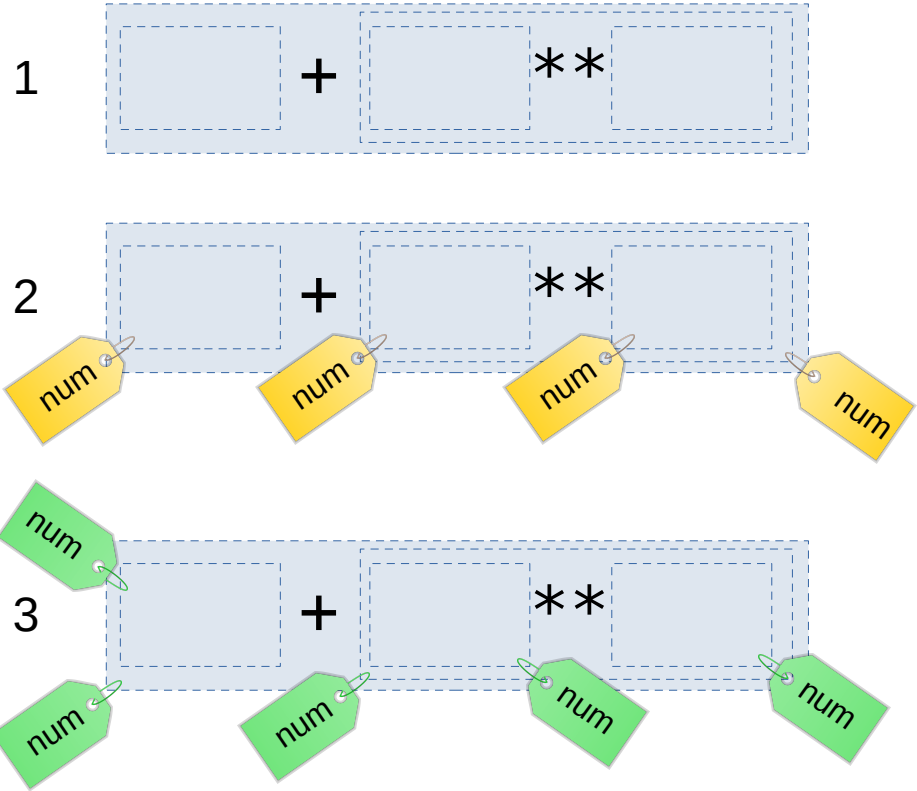
$$1 + 3.4 ** 2$$



Steps 1, 2 and 3:



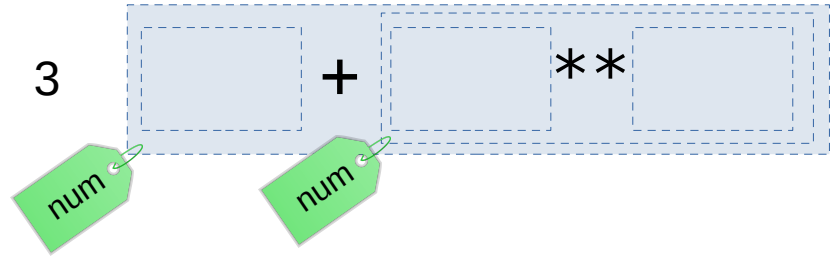
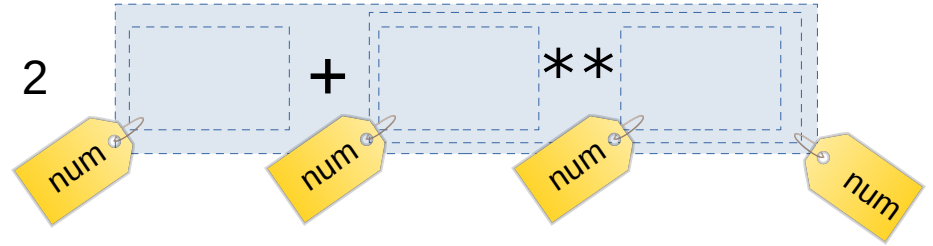
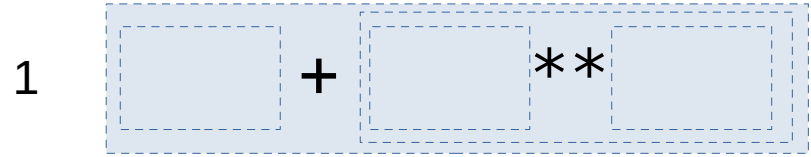
$1 + 3.4 ** 2$



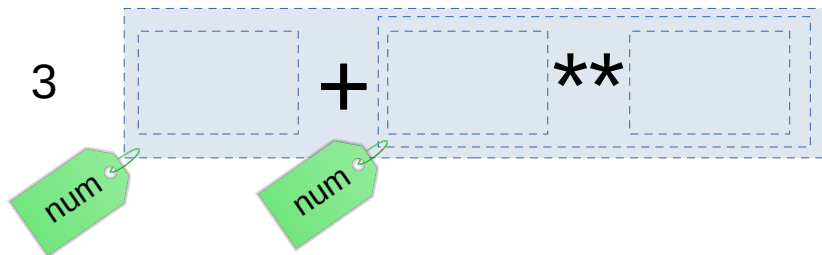
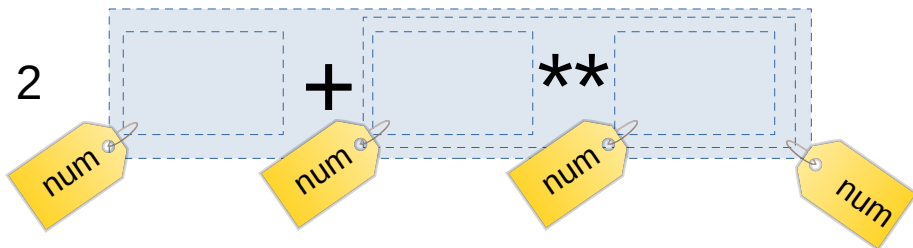
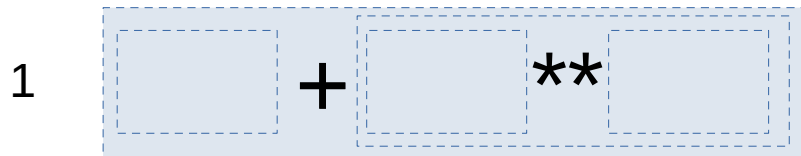
Steps 1, 2 and 3:

$$\boxed{1} + \boxed{3.4} ** \boxed{2}$$

$$1 + 3.4 ** 2$$

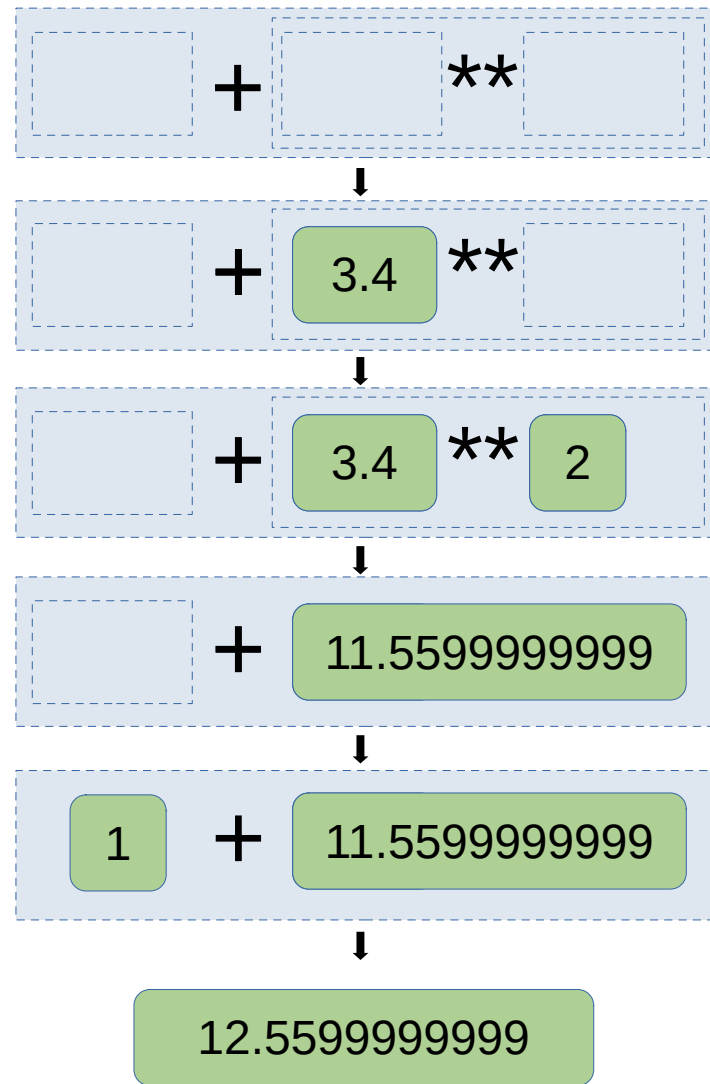
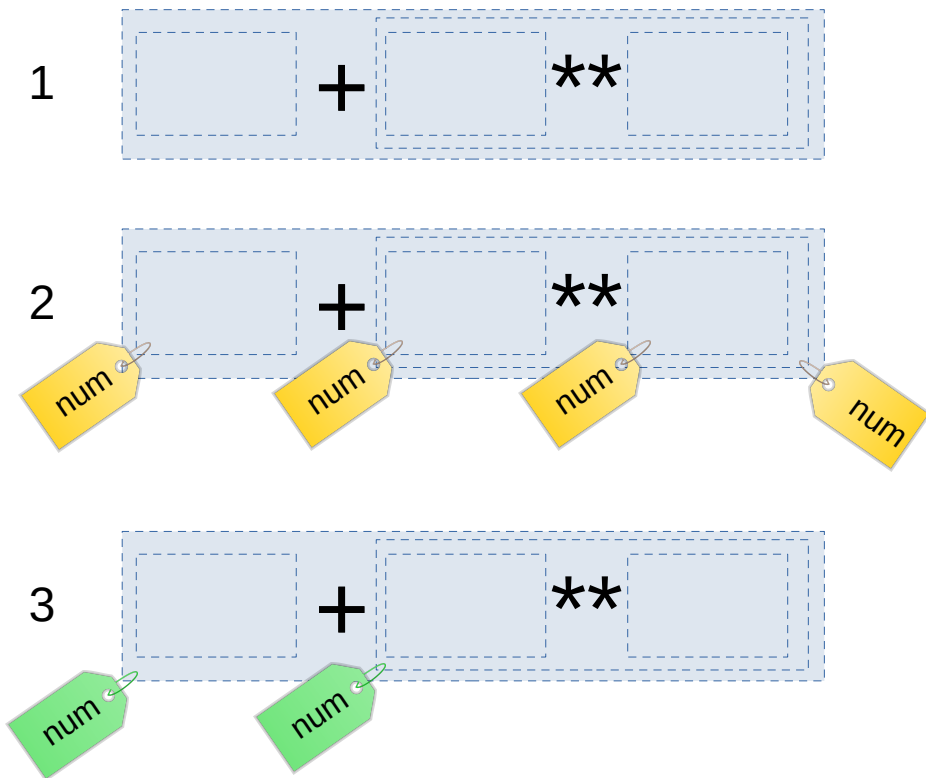


# Now we can compute





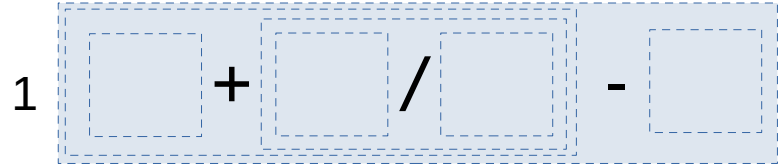
# Now we can compute



# Example 1

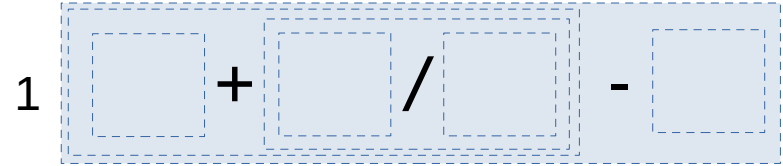
$$1 + 2 / 3 - 4$$

# Example 1

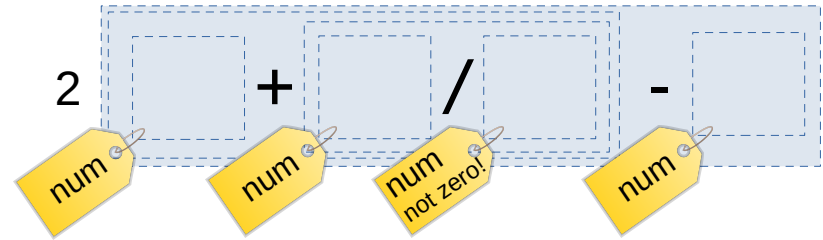


1 + 2 / 3 - 4

# Example 1

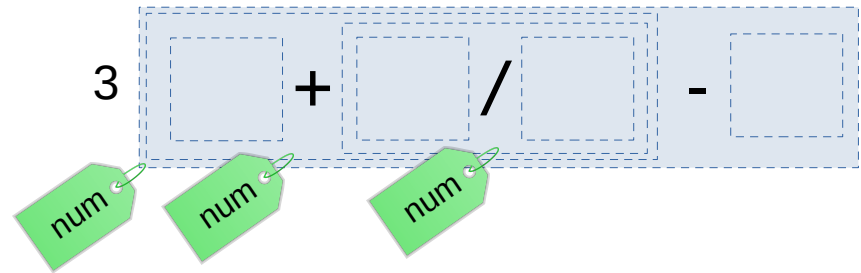
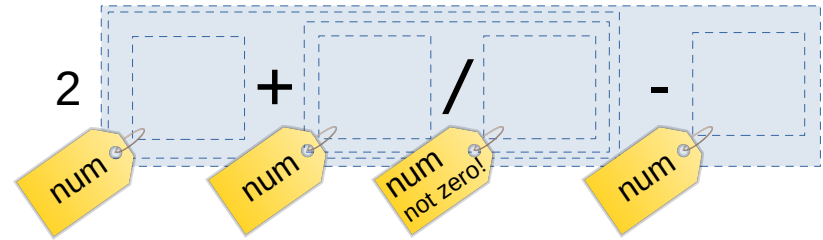
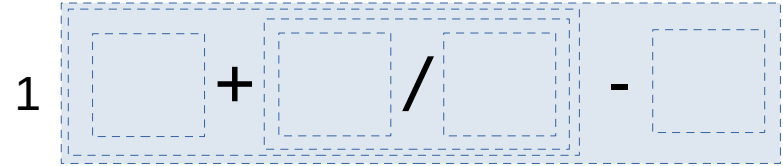


1 + 2 / 3 - 4



# Example 1

1 + 2 / 3 - 4



# Example 2

1 len( )


len("hello")

# Example 2

`len("hello")`

1 `len( )`

2 `len( )`



# Example 2

`len("hello")`

1 `len( )`

2 `len( )`

"count-able"

3 `len( )`

int

str



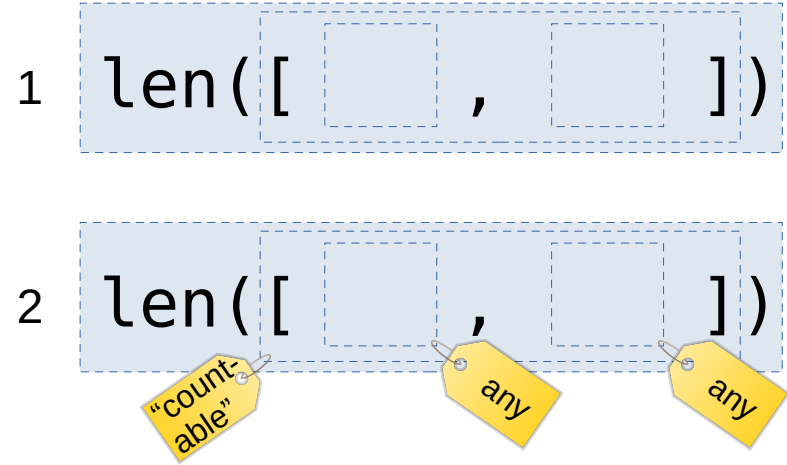
# Example 3

1 len([  ,  ])

len([22, "hi"])

# Example 3

```
len([22, "hi"])
```



# Example 3

`len([22, "hi"])`

1 `len([   ,   ])`

2 `len([   ,   ])`

"count-able" any any

3 `len([   ,   ])`

int list

# Example 4

```
stuff = (6,12,"hi")  
stuff[2:]
```

1 stuff[  :  ]

2 stuff[  :  ]

int or  
nothing

int or  
nothing

3 stuff[  :  ]

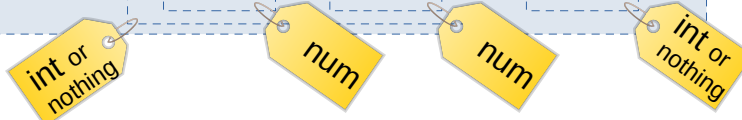
tuple

# Example 5


```
stuff = (6,12,"hi")  
stuff[1+1:]
```

1    stuff[   +   :   ]

2    stuff[   +   :   ]



3    stuff[   +   :   ]

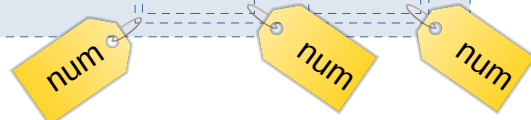


# Example 6

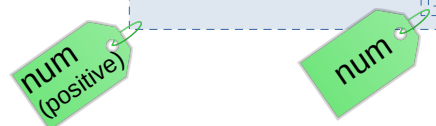
```
def power2(x):  
    return x**2  
power2(2+2)
```

1 power2(  +  )

2 power2(  +  )




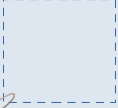
3 power2(  +  )




# Example 7

```
my_list = [1,1,"hi"]  
t = tuple(my_list)  
set(t)
```

1 set(  )

2 set(  )

col-  
lection

3 set(  )

set

# Example 7 cont.

```
my_list = [1,1,"hi"]  
set(tuple(my_list))
```

1 set(tuple( ))

2 set(tuple( ))

col-  
lection

col-  
lection

3 set(tuple( ))

set

tuple  
(a collection)

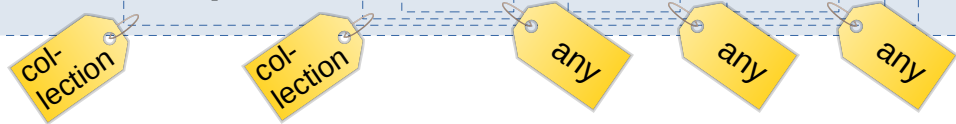


# Example 7 cont.

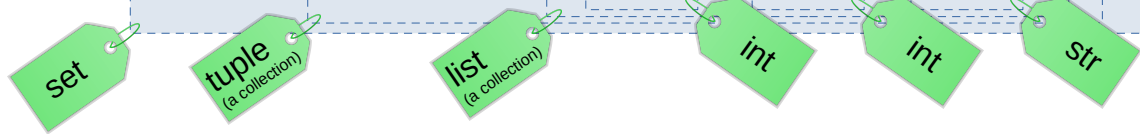
1 `set(tuple([ , , ]))`

`set(tuple([1,1,"hi"]))`

2 `set(tuple([ , , ]))`



3 `set(tuple([ , , ]))`



In the lecture, you've heard about tuples, lists and sets, and how to create them:

```
my_tuple = (1, "a", 1, 2)  # my_tuple is going to be (1, 'a', 1, 2)
my_list = [1, "a", 1, 2]   # my_list is going to be [1, 'a', 1, 2]
my_set = {1, "a", 1, 2}    # my_set is going to be {'a', 1, 2}
```

These data structures can also be created from each other using the `tuple`, `list` and `set` functions:

```
another_set = set(my_tuple)    # another_set is going to be {1, 'a', 2}
another_list = list(my_tuple)  # another_list is going to be [1, 'a', 1, 2]
another_tuple = tuple(my_set)  # another_tuple is going to be ('a', 1, 2)
```

With this new knowledge, implement a function `list_unique` which takes a **tuple** of arbitrary elements `elements` as the only parameter and returns a **list** containing only the unique elements contained in `elements`. The ordering of the elements in the return value does not matter.

**No 'for' loops needed...**

# Example 8

```
"hi".upper().endswith("I")
```

# Example 8

```
1 .upper().endswith()
```

```
"hi".upper().endswith("I")
```

# Example 8

"hi".upper().endswith("I")

1  .upper().endswith()

2  .upper().endswith()

3  .upper().endswith()

str

str

str

bool

str

# Example 9

```
tuple(set(["bob"]))[0].capitalize()
```

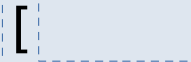

# Example 9

```
tuple(set(["bob"]))[0].capitalize()
```

```
1 tuple(set([ ]))[ ].capitalize()
```

# Example 9

```
tuple(set(["bob"]))[0].capitalize()
```

1    tuple(set([  ])) [  ].capitalize()

2    tuple(set([  ])) [  ].capitalize()

str?

int



# Example 9

```
tuple(set(["bob"]))[0].capitalize()
```

1 tuple(set([ ]))[ ].capitalize()

2 tuple(set([ ]))[ ].capitalize()

str?

int

3 tuple(set([ ]))[ ].capitalize()

tuple

set

list

str

str

# You already know all of this!

Now you're just learning some new syntax and functionality

$$\sum_{i=3}^6 i^2 \approx \text{sum}([i**2 \text{ for } i \text{ in range}(3,7)])$$

$$\sqrt{\frac{1}{2}}$$

