Lecture 2: Data Structures

Chirag Sakhuja

Overview

- Recap
- Strings
- Lists
- Tuples
- Sets
- Dictionaries

Python arithmetic operators

+	Addition
-	Subtraction
*	Multiplication
/	Division (floating point)
//	Division (integer)
%	Modulus
**	Exponentiation

Boolean operators

`==`	Equal
`!=`	Not equal
`>`	Greater than
`>=`	Greater than or equal
`<`	Less than
`<=`	Less than or equal
`not`	Logical NOT
`and`	Logical AND
`or`	Logical OR

Strings

```
str1 = "Hello"     # => 'Hello'
str2 = 'Hello'     # => 'Hello'
str1 == str2       # => True
str1 + ', world!' # => 'Hello, world!'
```

No difference between ' and "

The in operator

```
x = 'Chirag'
'a' in x                                    # => True
'b' in x                                    # => False
# equivalent to x == 'Chirag'
('Chirag' in x) and (x in 'Chirag')         # => True
```

If statements

```
if cond1:
    print('cond1 was True')
elif cond2:
    print('cond1 was False, but cond2 was True')
else:
    print('cond1 and cond2 were False')
```

For loops

```
for item in iterable:
    # code to handle item


l = '123'
for x in l:
    print(x)    # => 1
                     2
                     3
```

While loops

```
while cond:
    # execute this until cond is False

# loop until user types in 'q'
while input() != 'q':
    # do something
```

Overview

- Recap
- Strings
- Lists
- Tuples
- Sets
- Dictionaries

Indexing

```
          0  1  2  3  4  5
name = 'Chirag'
         -6 -5 -4 -3 -2 -1
```

Slicing

```
string = 'abcdefg'
string[0:2]                 # => 'ab'
string[0:-1]                # => 'abcdef'
string[0:-1:2]              # => 'ace'
string[:4]                  # => 'abcd'
string[:4:2]                # => 'ac'
string[::-1]                # => 'gfedcba'
string[4:1:-1]              # => 'edc'
string[-1::-1]              # => 'gfedcba'
```

String operations

```
s = 'Hello, world! '
'world' in s                    # => True
len(s)                          # => 14
s.lower()                       # => 'hello, world! '
s.upper()                       # => 'HELLO, WORLD! '
s.strip()                       # => 'Hello, world!'
s.find('world')                 # => 7 (-1 if not found)
s.replace('world', 'earth') # => 'Hello, earth! '
s.replace('o', 'u')             # => 'Hellu, wurld! '
```

String formatting

```
str1 = 'Hello'
str2 = 'world'
'{}, {}!'.format(str1, str2)
# => 'Hello, world!'
'{0}, {1}, {0}'.format('first', 'second')
# => 'first, second, first'
'{:.2f}'.format(2.71828)
# => 2.72
```

Overview

- Recap
- Strings
- Lists
- Tuples
- Sets
- Dictionaries

Lists can contain anything, thanks duck typing!

```
l = [1, 2, 3]
l = [1, 2, 'three']
l = [1, 2, [3, 4], [5]]

l.append('six')     # => [1, 2, [3, 4], [5], 'six']
```

Lists denoted by square brackets

Duck typing, again

"If it walks like a duck and it quacks like a duck, then it must be a duck."

```
'a' + ('bc' * 2)      # => 'abcbc'
[1] + ([2, 3] * 2)    # => [1, 2, 3, 2, 3]
```

Slicing, again

```
row0 = [0, 1, 2]
row1 = [3, 4, 5]
row2 = [6, 7, 8]
mat = [row0, row1, row2]

mat[-1]          # => [6, 7, 8]
mat[1:]          # => [[3, 4, 5], [6, 7, 8]]
mat[1:][1:]      # => [[6, 7, 8]]
```

The in operator, again

```
jagged = [[0], [1, 2], [3, 4, 5]]
0 in jagged                                  # => False
[0] in jagged                                # => True
[1, 2] in jagged                             # => True
for row in jagged:
    print('{} ({})'.format(row, len(row)))
    # => [0] (1)
    # => [1, 2] (2)
    # => [3, 4, 5] (3)
```

List operations

```
l = [1, 2, 3]
l.append(4)       # => [1, 2, 3, 4]
l.append(2)       # => [1, 2, 3, 4, 2]
l.remove(2)       # => [1, 3, 4, 2]
l.sort()          # => [1, 2, 3, 4]
l.pop()           # => [1, 2, 3]
l.pop(1)          # => [1, 3]
l.reverse()       # => [3, 1]
l.clear()         # => []
```

Converting between strings and lists

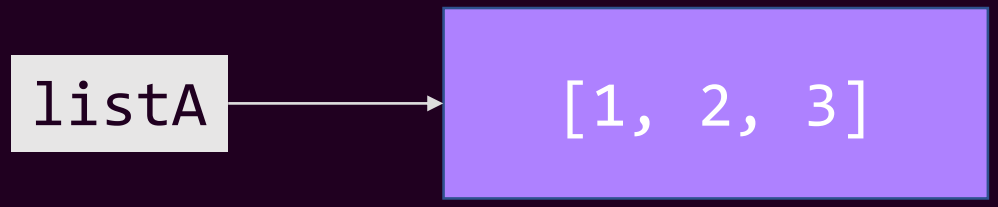
```
s = '1, 2, 3, 4, 5'     # => '1, 2, 3, 4, 5'
l = s.split(', ')       # => [1, 2, 3, 4, 5]
l.remove(3)             # => [1, 2, 4, 5]
s = '; '.join(l)        # => '1; 2; 4; 5'
```

"Memory management"

- Python manages memory for you, yay!
- Variables are pointers to memory
- When you "copy" a variable, it's just adding another reference to the data

Digging a little deeper

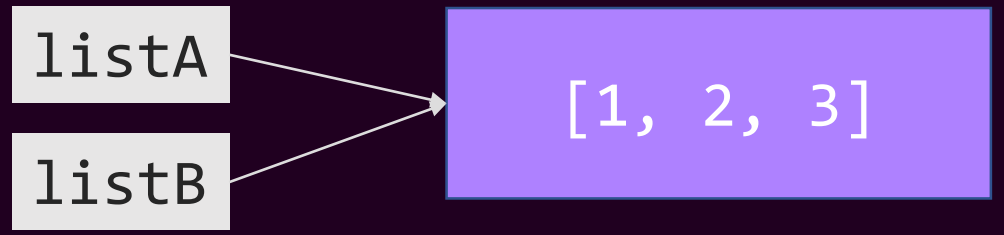
```
listA = [1, 2, 3]
```



Digging a little deeper

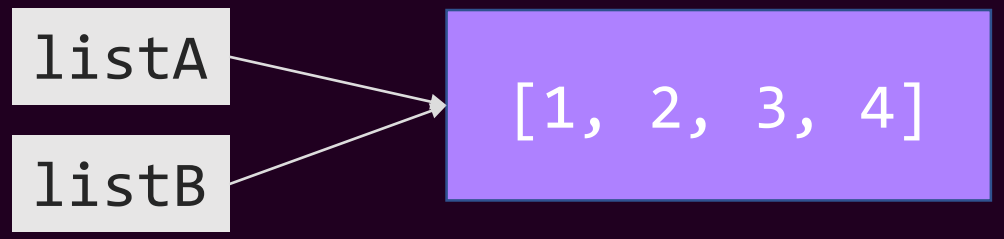
```
listA = [1, 2, 3]
listB = listA
```

Adds another reference



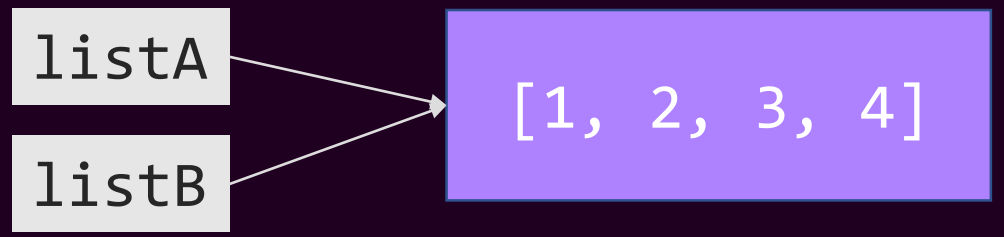
Digging a little deeper

```
listA = [1, 2, 3]
listB = listA
listB.append(4)
```



Digging a little deeper

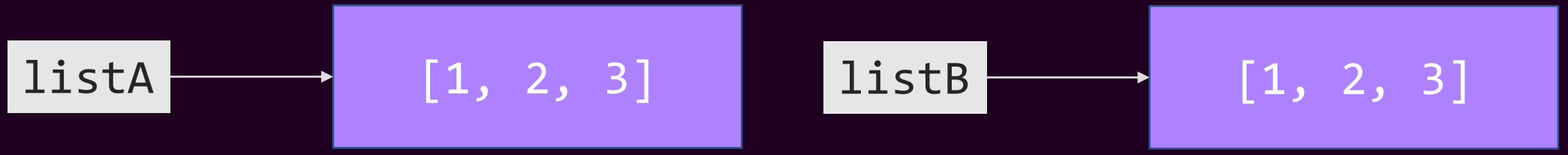
```
listA = [1, 2, 3]
listB = listA
listB.append(4)
print(listA)    # => [1, 2, 3, 4]
```



Digging a little deeper

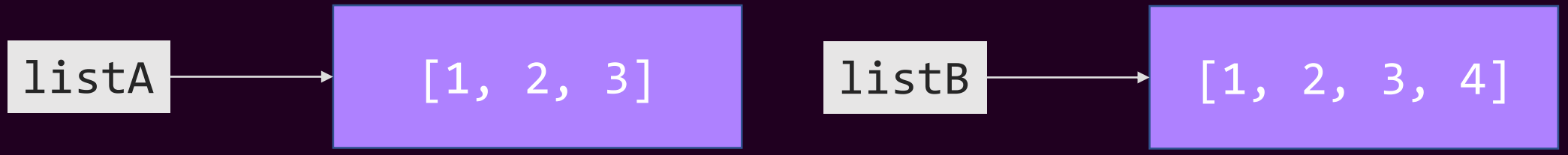
```
listA = [1, 2, 3]
listB = listA.copy()
```

Creates a copy



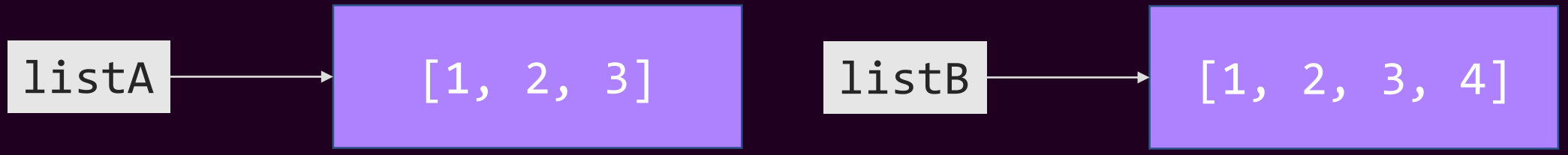
Digging a little deeper

```
listA = [1, 2, 3]
listB = listA.copy()
listB.append(4)
```



Digging a little deeper

```
listA = [1, 2, 3]
listB = listA.copy()
listB.append(4)
print(listA)    # => [1, 2, 3]
```



Overview

- Recap
- Strings
- Lists
- Tuples
- Sets
- Dictionaries

Tuples

- Tuples function similar to lists, but are used differently
- Typically use tuples to tie together related values
  - Similar to how structs are used in C
  - Python does have classes; we'll get to them later
- Tuples are immutable

Tuples

```
tup = ('Tuesday', 2017, 1, 30)
# normal operations
len(tup)                # => 4
tup[0]                  # => 'Tuesday'
tup[1:]                 # => (2017, 1, 30)
'Tuesday' in tup        # => True
# immutable
tup[1] = 2018           # => TypeError!
```

Tuples denoted by parenthesis

Tuple packing/unpacking

```
tup = 1, 2
```

Comma-separated r-values pack into a tuple

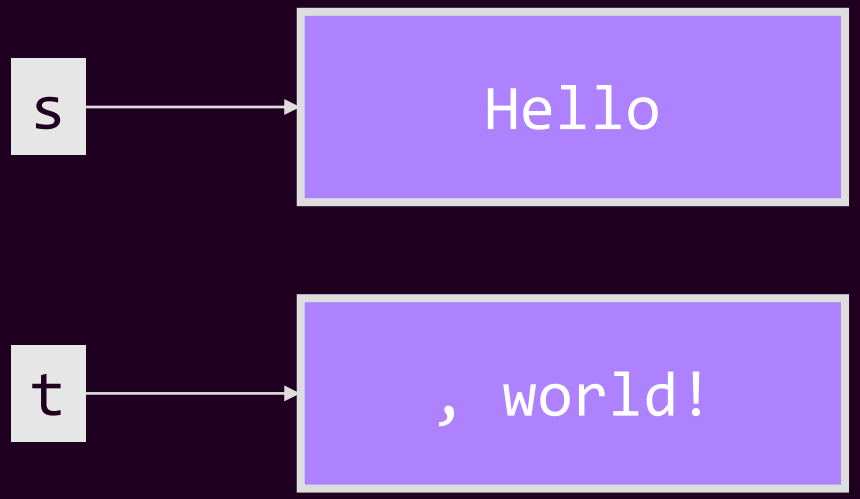
```
a, b = tup
```

Comma-separated l-values unpack a tuple

```
print(a)        # => 1
print(b)        # => 2
a, b = b, a
print(a)        # => 2
print(b)        # => 1
```

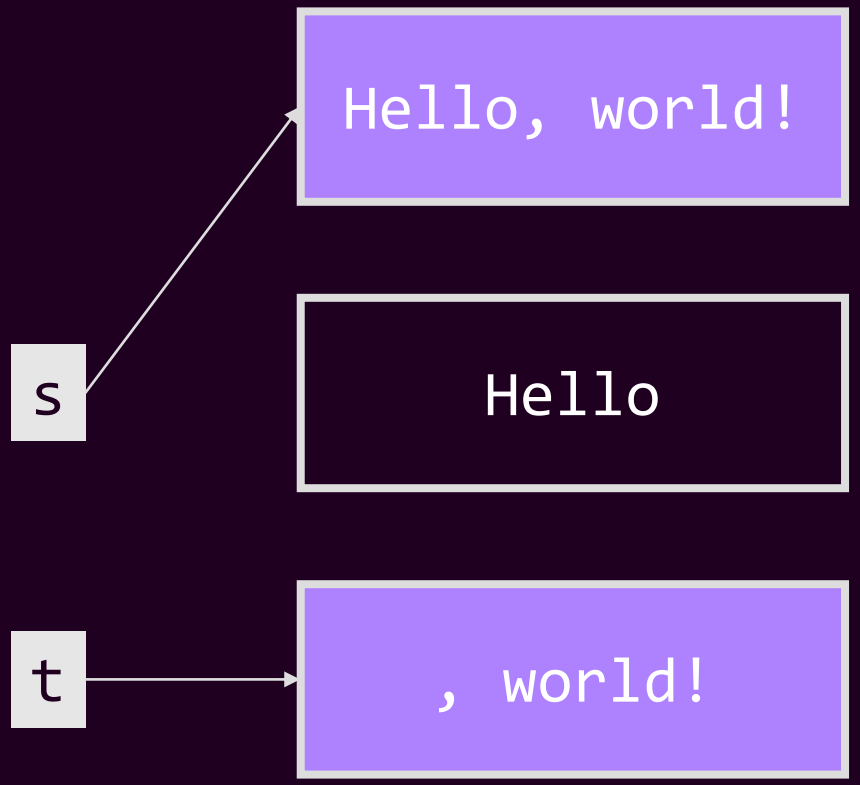
What does "immutable" mean?

```
s = 'Hello
t = ', world!'
```



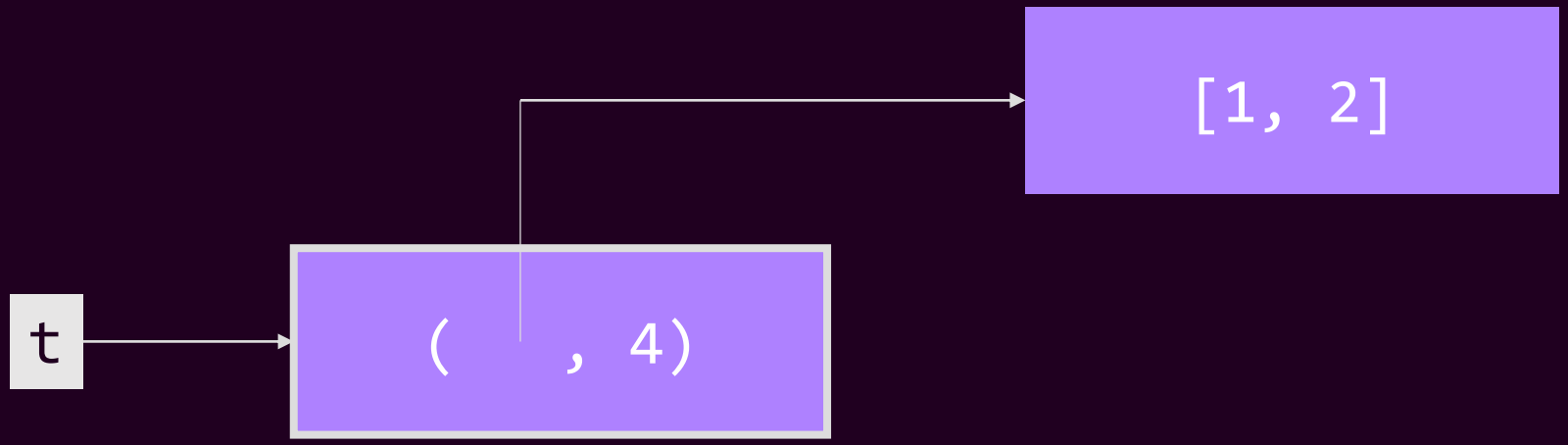
What does "immutable" mean?

```
s = 'Hello
t = ', world!'
s = s + t
```



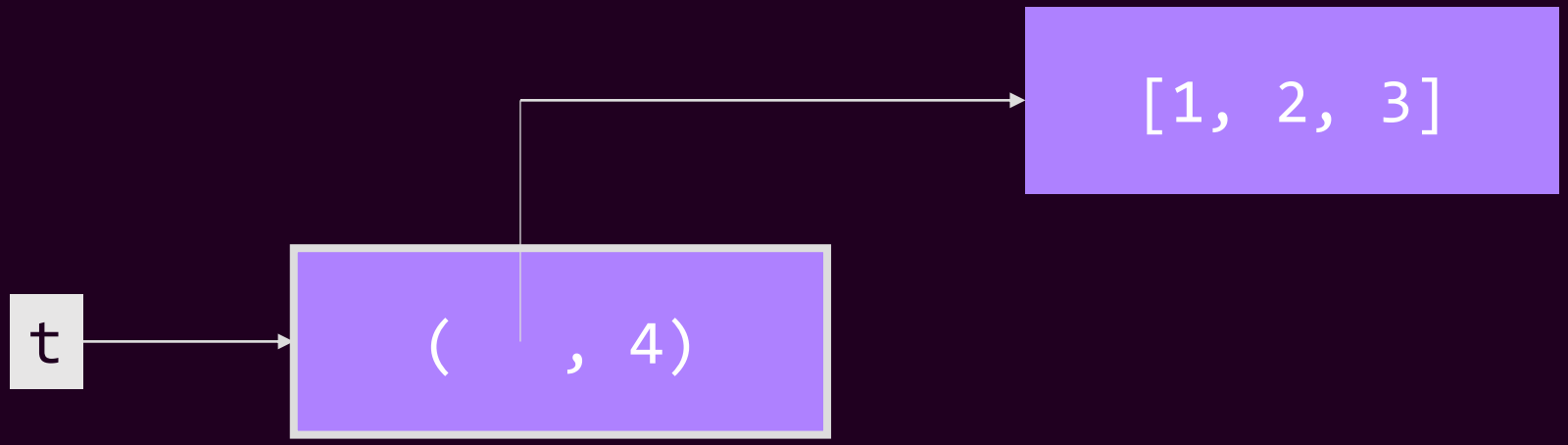
What does "immutable" mean?

```
tup = ([1, 2], 4)
```



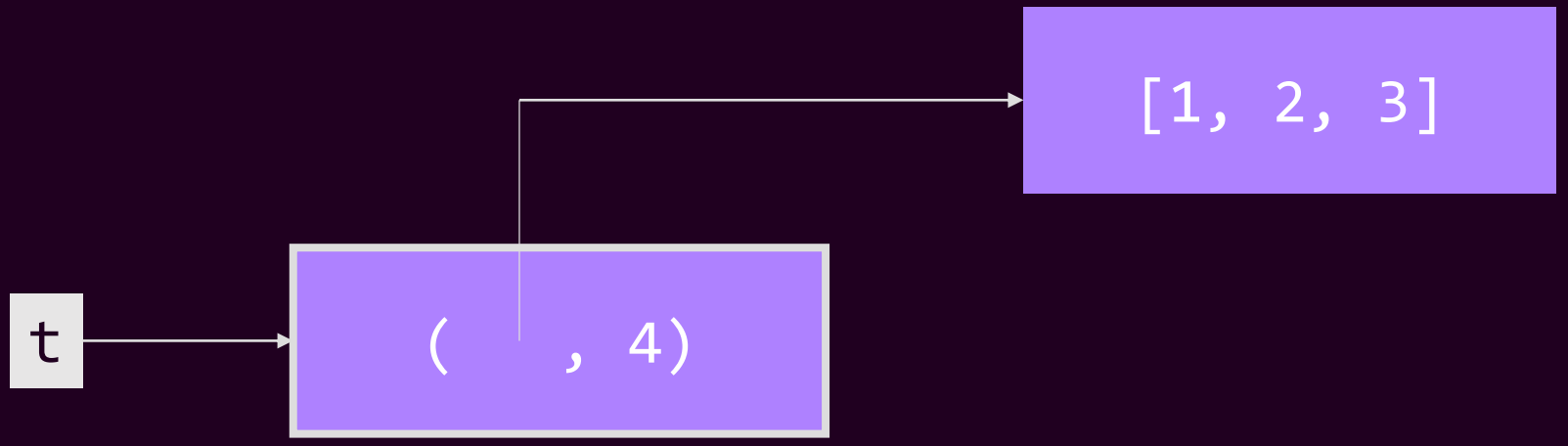
What does "immutable" mean?

```
tup = ([1, 2], 4)
tup[0].append(3)
```



What does "immutable" mean?

```
tup = ([1, 2], 4)
tup[0].append(3)
tup[1] = 5      # => TypeError!
```



Overview

- Recap
- Strings
- Lists
- Tuples
- Sets
- Dictionaries

Sets

- Sets are also like lists, but have a differently underlying data structure
- Just like in mathematics, a set cannot contain duplicates of an item
- O(1) membership test!

Sets can contain anything, thanks duck typing!

```
s = {1, 2, 3}
s = {1, 2, 'three'}
s = set([1, 2, 3, 3])
s = set('Hello')
s[0]
```

```
# => {1, 2, 3}
# => {'o', 'e', 'H', 'l'}
# => TypeError
```

Sets denoted by curly braces

Set operations

```
s = set('mississippi')  # => {'p', 'm', 's', 'i'}

len(s)                  # => 4
s.add('q')              # => {'i', 'q', 'm', 's', 'p'}
s.remove('s')           # => {'i', 'q', 'm', 'p'}
len(s)                  # => 4

'm' in s                # => True
```

Set operations

```
s1 = set('Hello')     # => {'e', 'l', 'H', 'o'}
s2 = set('world')     # => {'l', 'd', 'o', 'r', 'w'}
# difference
s1 - s2               # => {'H', 'e'}
s2 - s1               # => {'w', 'r', 'd'}
# union
s1 | s2               # => {'H', 'e', 'l', 'w', 'd', 'o', 'r'}
# intersection
s1 & s2               # => {'o', 'l'}
# symmetric difference
s1 ^ s2               # => {'H', 'e', 'w', 'r', 'd'}
```

Overview

- Recap
- Strings
- Lists
- Tuples
- Sets
- Dictionaries

Dictionaries

- Sometimes called a map or dict
- Maps a set of keys to values of any type
- Think of it like a lookup table

Dicts

```
a = dict(one=1, two=2)
b = {'one': 1, 'two': 2}
a == b                          # => True

empty = {}
```

Dict denoted by curly braces

Empty curly braces create dict, not set

Dict operations

```
grades = {'Chirag': [93, 87], 'Cassidy': [100, 94]}

grades['Chirag']       # => [93, 87]
grades['John']         # => KeyError

grades['Chirag'][0] = 100 # => {'Chirag': [100, 87],
                                'Cassidy': [100, 94}
```

Dict operations

```
grades = {'Chirag': [93, 87], 'Cassidy': [100, 94]}

len(grades)             # => 2
del grades['Chirag']    # => {'Cassidy': [100, 94]}
len(grades)             # => 1

grades.keys()     # => dict_keys(['Cassidy'])
grades.values()   # => dict_values([[100, 94]])
grades.items()    # => dict_items([('Cassidy', [100, 94])])

('Cassidy', [100, 94]) in grades.items()  # => True
```

Iterating over a dict

```
grades = {'Chirag': [93, 87], 'Cassidy': [100, 94]}

for name in grades:
    print(name)     # => 'Chirag'
                    # => 'Cassidy'
for grade in grades.values():
    print(grade)    # => [93, 87]
                    # => [100, 94]
```

Implied `.keys()`

Iterating over a dict

```
grades = {'Chirag': [93, 87], 'Cassidy': [100, 94]}

for name, grade in grades.items():
    print('{}: {}'.format(name, grade))
    # => Chirag: [93, 87]
    # => Cassidy: [100, 94]
```

Poll

- Interesting?
- Too fast? Too slow?
- Insert practical examples?
- Insert "quizzes"?
- Write programs in real time?

I'll send a "mid-semester" survey out after next lecture

Key insights

- Python has several data structures as first-class citizens of the language
  - Pick the right one for the task
  - Easy to convert between data structures
- There is **a lot** we didn't cover
  - Use the documentation! https://docs.python.org/3/
- New syntax, potentially new data structures, a different way of thinking – it can be overwhelming, but use features as needed and build your knowledge with experience

A reminder

PRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRAC
TICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICE
PRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRAC
TICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICE
PRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRAC
TICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICE
PRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRAC
TICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICE
PRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRAC
TICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICE
PRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRACTICEPRAC

Credits

You may notice a striking similarity between my slides and the Stanford Python course…that's not a coincidence

http://stanfordpython.com