Tuples

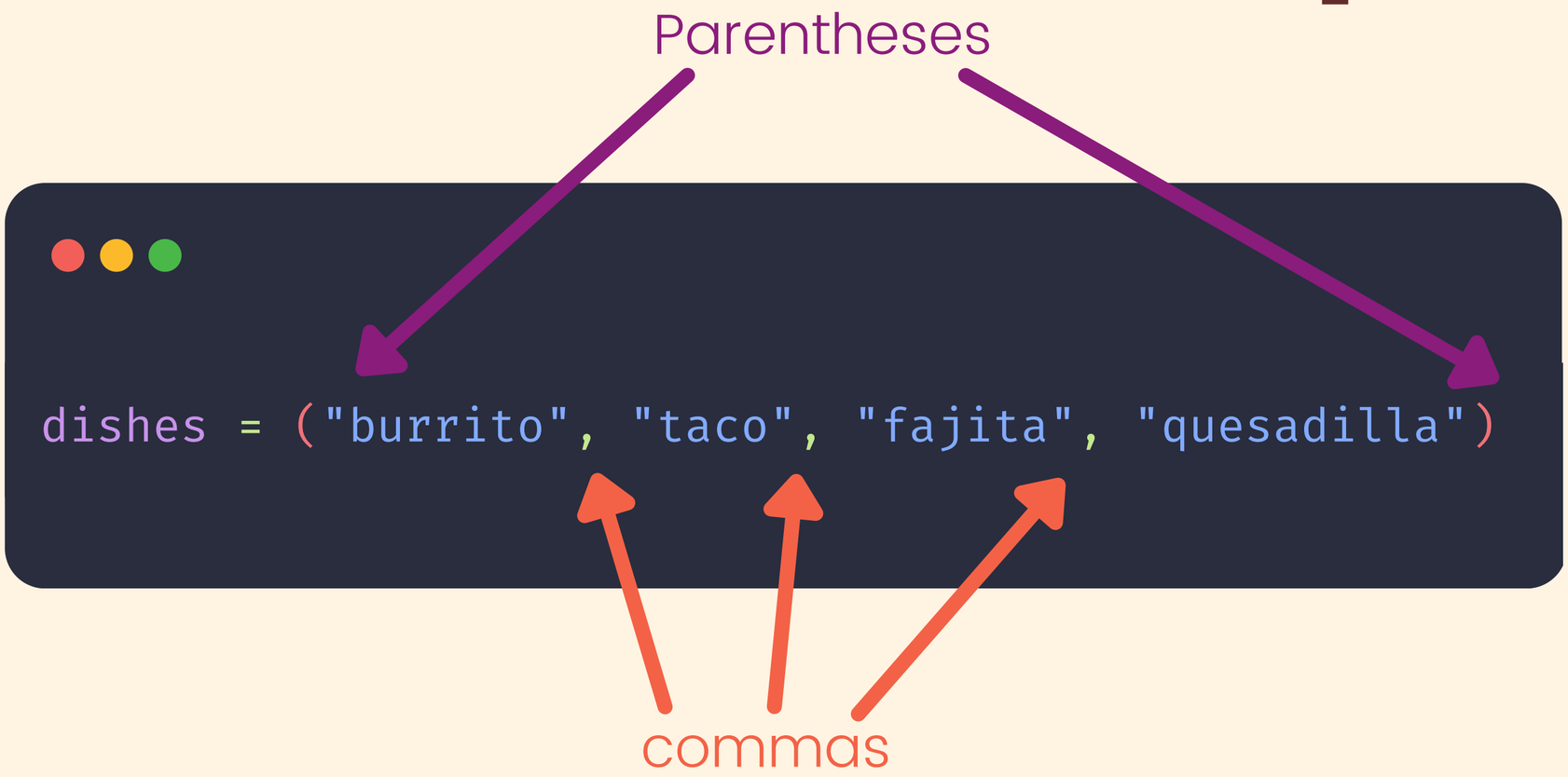
Tuples

Immutable, ordered collections

tuples

- Like lists, tuples are ordered, indexed collections
- Unlike lists, **tuples are immutable.** They cannot change once created

Create a Tuple



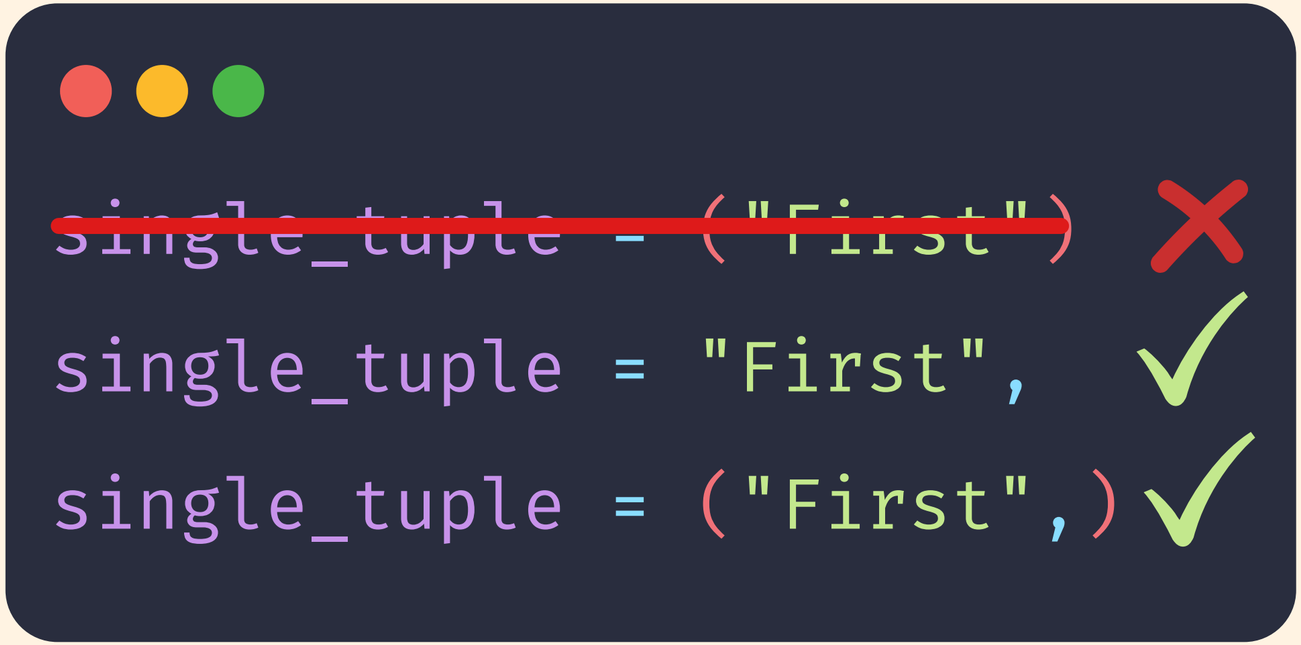
Empty Tuple

```
empty_tuple = ()
empty_tuple = tuple()
```

1 Item Tuple

```
single_tuple = ("First")
single_tuple = "First",
single_tuple = ("First",)
```

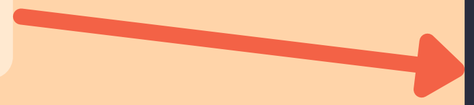
1 Item Tuple



Action

Code

Access



```
> dishes = ("burrito", "taco", "fajita", "quesadilla")

> dishes[2]
  'fajita'
```

Action

Code

Access

Slice

```
> dishes = ("burrito", "taco", "fajita", "quesadilla")

> dishes[2]
  'fajita'
> dishes[1:3]
  ('taco', 'fajita')
```

Action

Code

Access

Slice

Index

```
> dishes = ("burrito", "taco", "fajita", "quesadilla")

> dishes[2]
  'fajita'
> dishes[1:3]
  ('taco', 'fajita')
> dishes.index("taco")
  1
```

Action

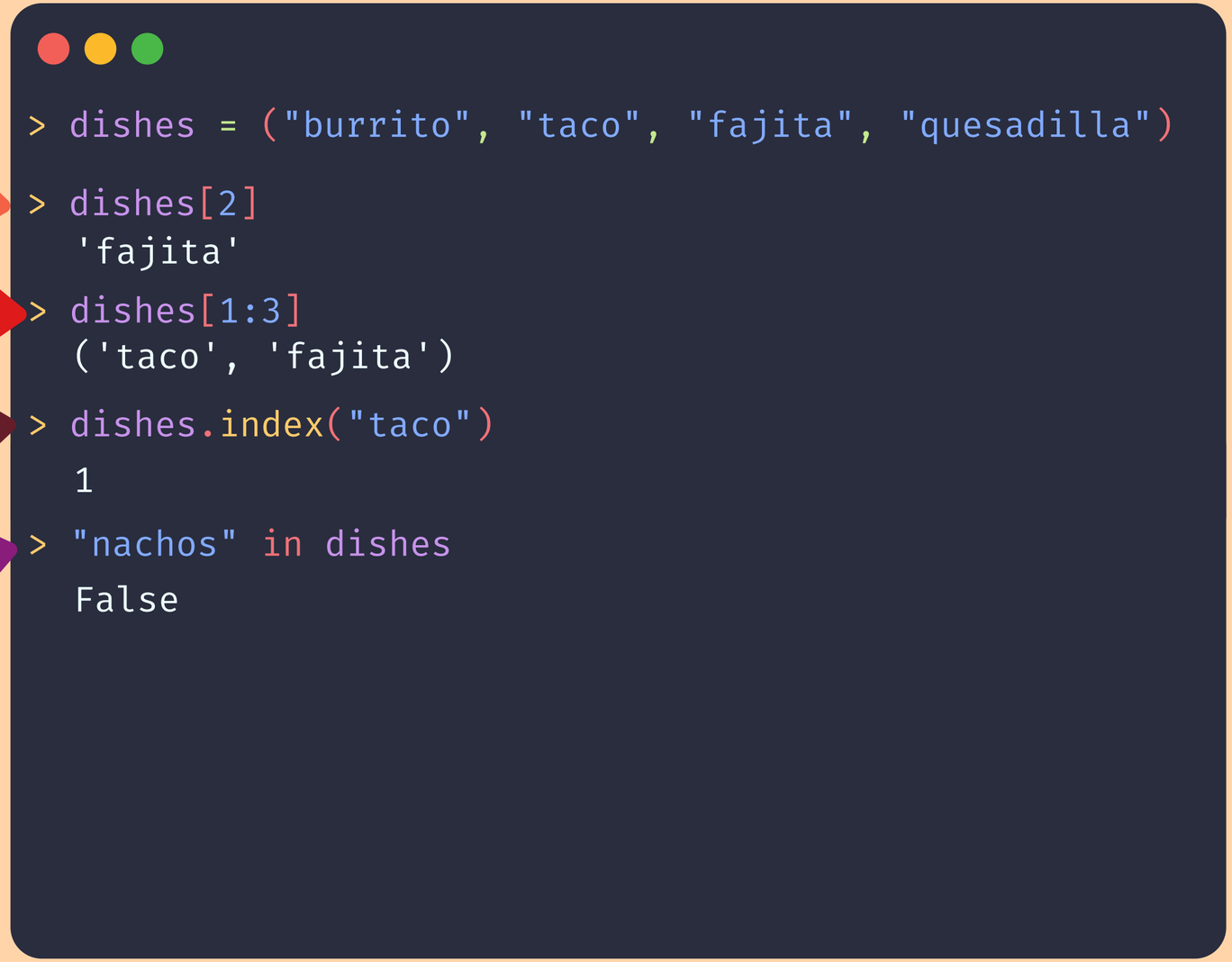
Code

Access

Slice

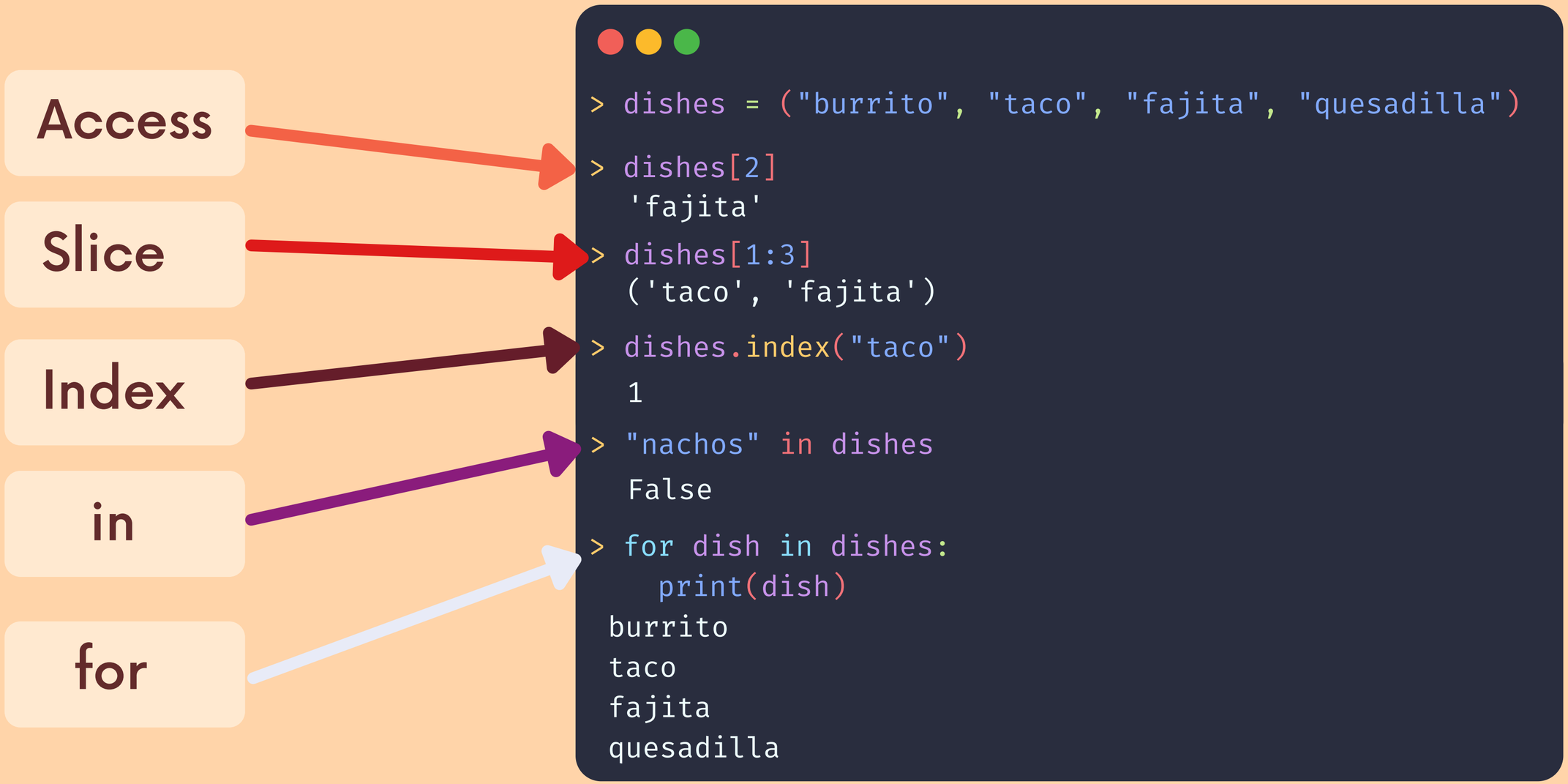
Index

in



Action

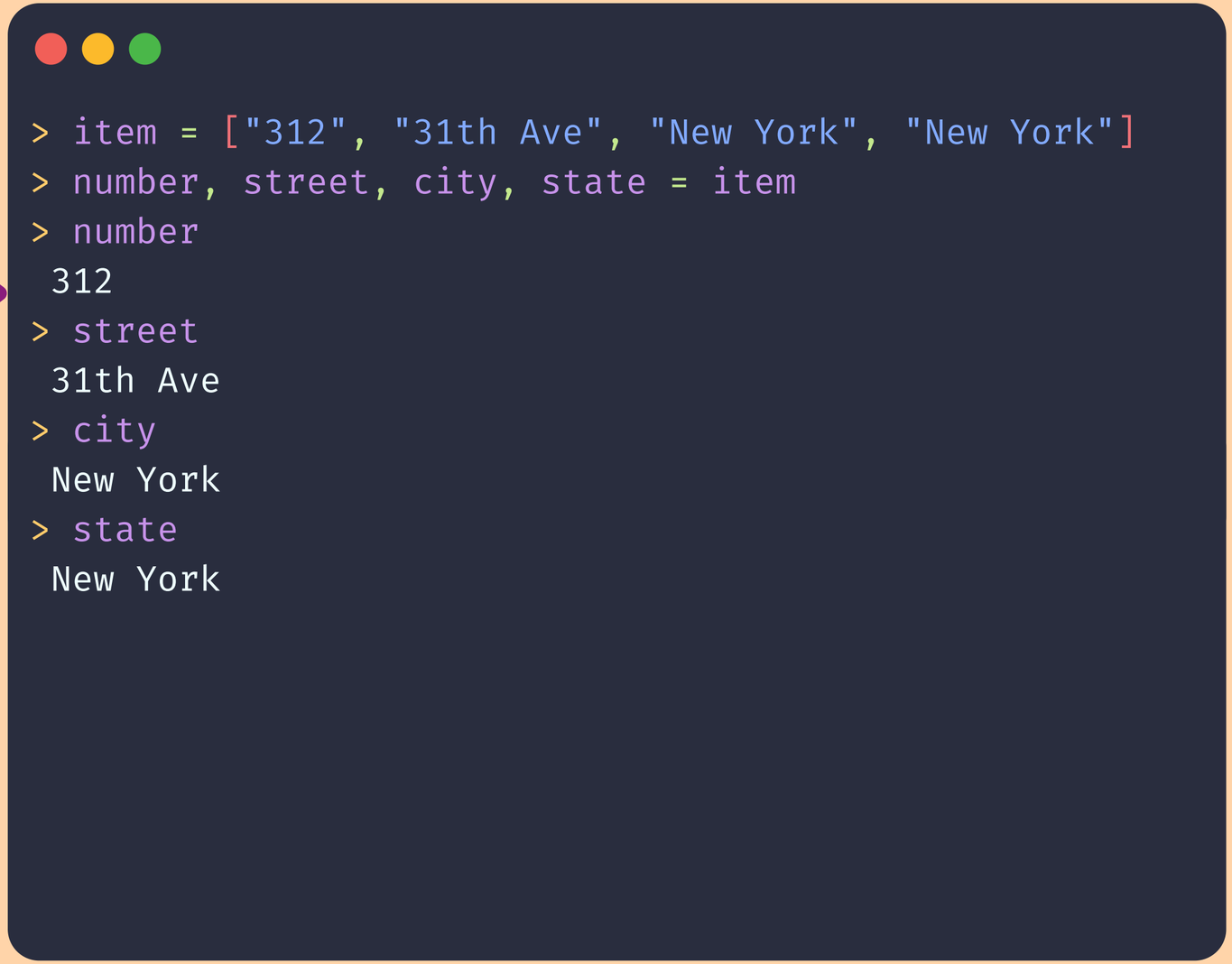
Code


Access
Slice
Index
in
for
> dishes = ("burrito", "taco", "fajita", "quesadilla")
> dishes[2]
'fajita'
> dishes[1:3]
('taco', 'fajita')
> dishes.index("taco")
1
> "nachos" in dishes
False
> for dish in dishes:
print(dish)
burrito
taco
fajita
quesadilla

Action

Code

Unpacking

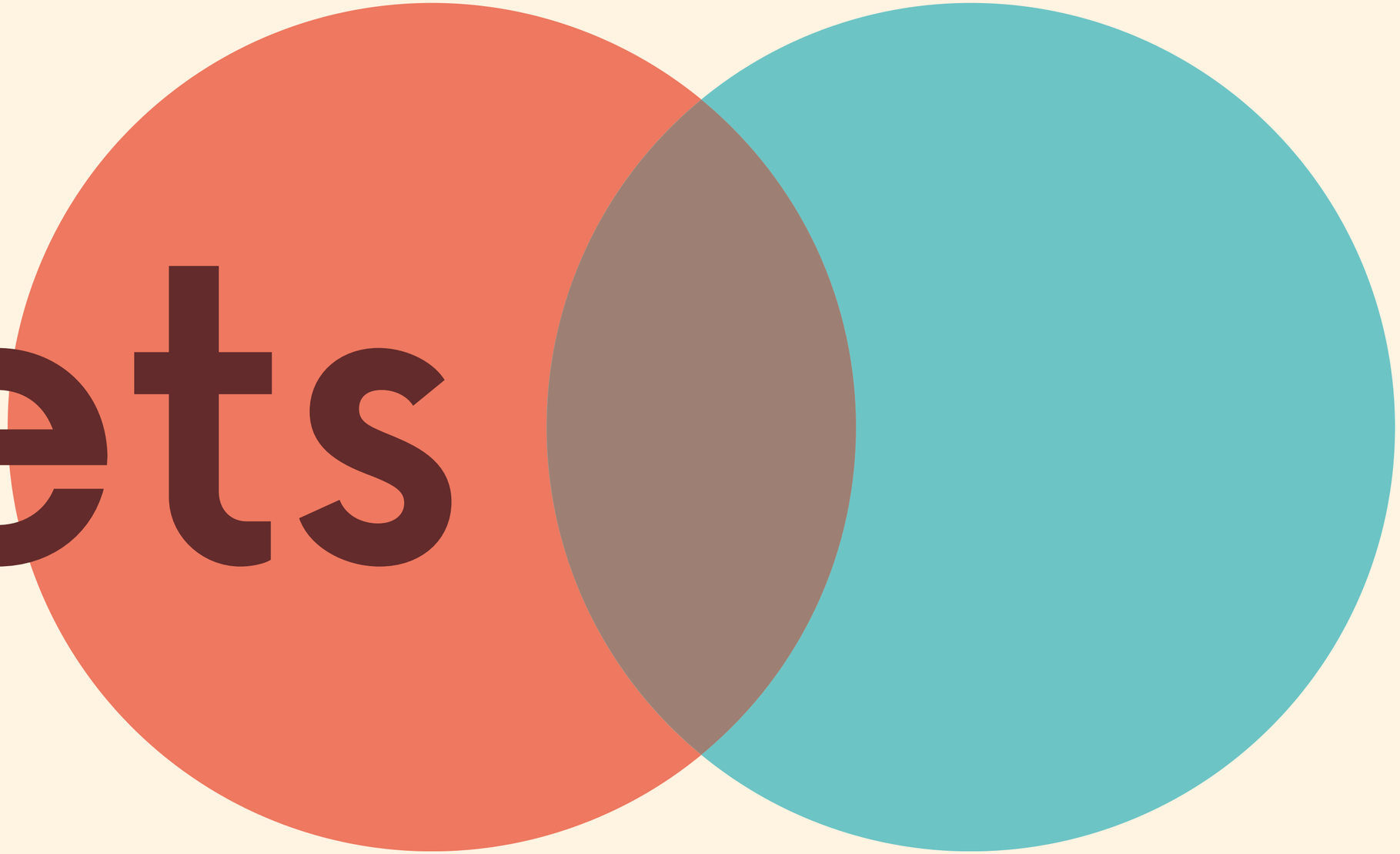


```
> item = ["312", "31th Ave", "New York", "New York"]
> number, street, city, state = item
> number
 312
> street
 31th Ave
> city
 New York
> state
 New York
```

why use tuples?

- they are more efficient than lists
- use them for data that shouldn't change
- some methods return them like dict.items()
- they can be used as keys in a dictionary

Sets



Sets

Unordered collections
with **no duplicate elements**

Sets

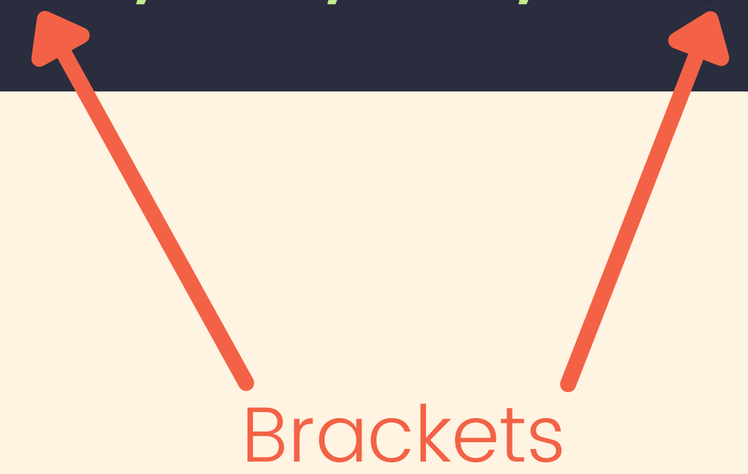
Only **immutable** elements!

We can...

- Add and delete elements
- Iterate over elements
- Check to see if element is in a set
- Use set operators: union, intersection, etc.

Creating Sets

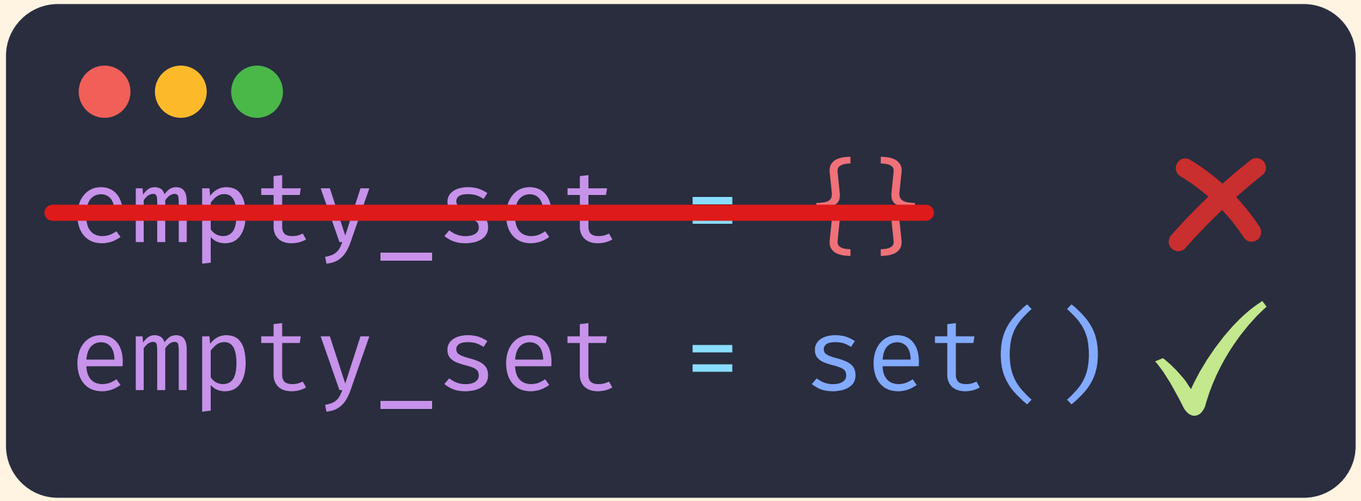
```
evens = {2, 4, 6, 8}
```



Empty Set

```
empty_set = {}
empty_set = set()
```

Empty Set



add()

```
> even = {2, 4, 6, 8}
> even.add(12)
> even
{2, 4, 12, 6, 8}
```

Adds a single value to a set. No duplicates in sets!

remove()

```
> langs = {"Python", "C", "JavaScript"}
> lang.remove("C")
> langs

{Python, JavaScript}
```

Removes a single value from a set

discard()

```
> langs = {"Python", "C", "JavaScript"}
> lang.discard("C")
> langs

{Python, JavaScript}
```

Like remove() but won't throw error for missing value

clear()

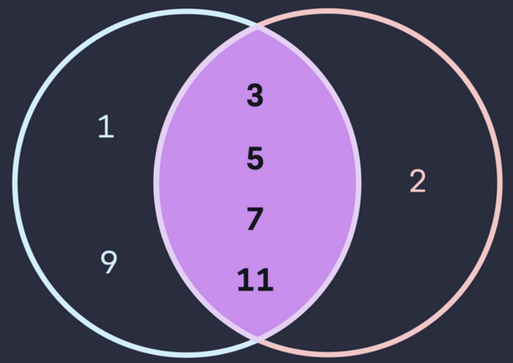
```
> langs = {"Python", "C", "JavaScript"}
> langs.clear()
> langs
set()
```

Empties out a set

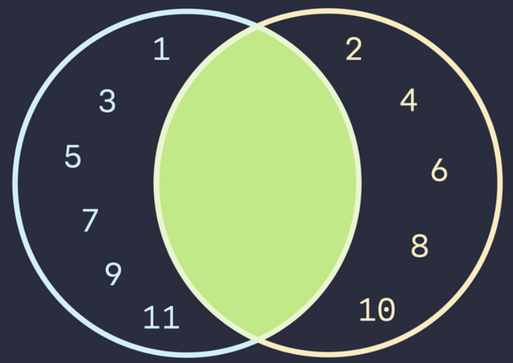
Intersection

returns new set with members common to left and right

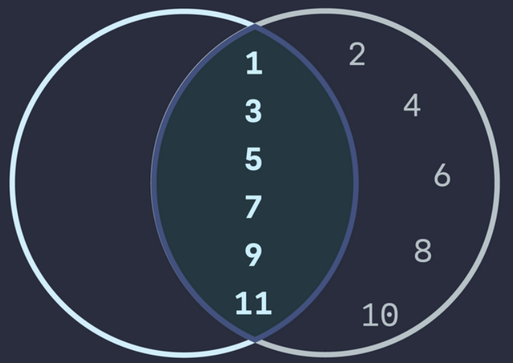
left & right



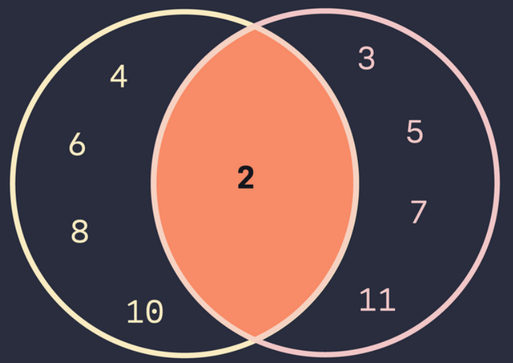

```
> set_odd & set_prime
{3, 5, 7, 11}
```




```
> set_odd & set_even
{}
```




```
> set_odd & set_all
{1, 3, 5, 7, 9, 11}
```

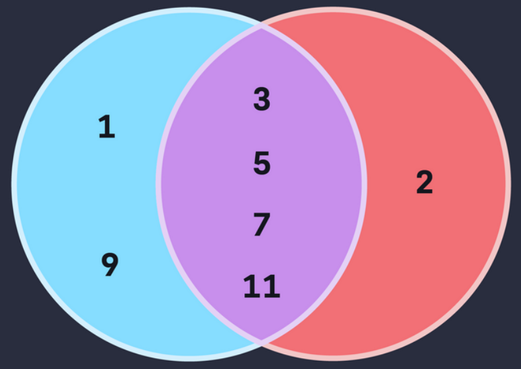



```
> set_even & set_prime
{2}
```

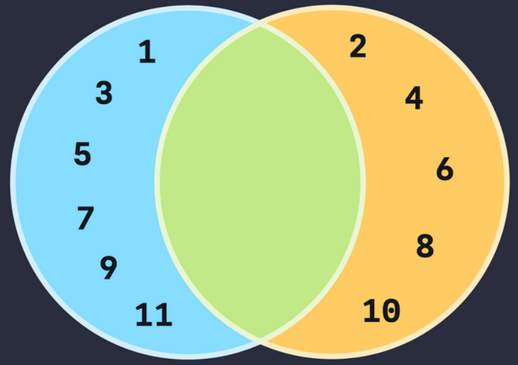
Union

returns new set with
members from left and right

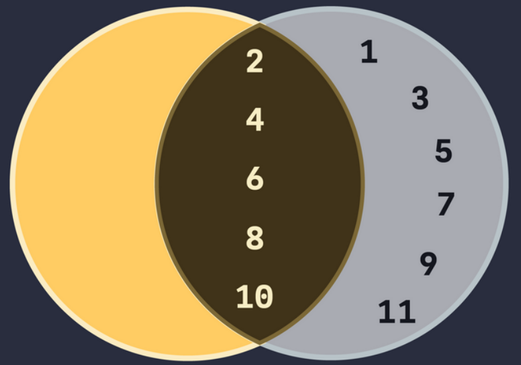
`left | right`



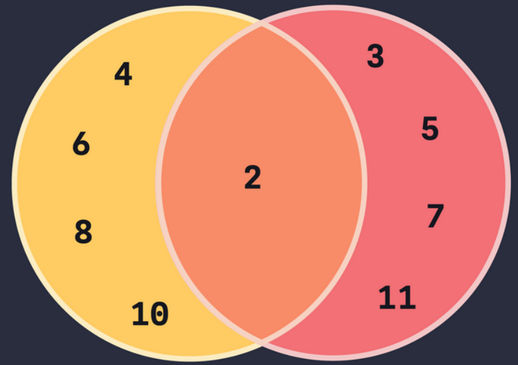

```
> set_odd | set_prime
{1, 2, 3, 5, 7, 9, 11}
```




```
> set_odd | set_even
{1, 2, 3, 4, 5, 6,
7, 8, 9, 10, 11}
```




```
> set_even | set_all
{1, 2, 3, 4, 5, 6,
7, 8, 9, 10, 11}
```

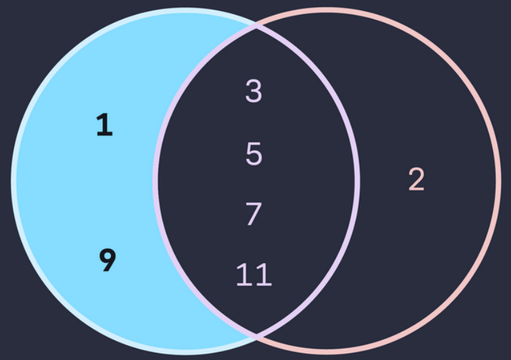



```
> set_even | set_prime
{2, 3, 4, 5, 6, 7, 8,
10, 11}
```

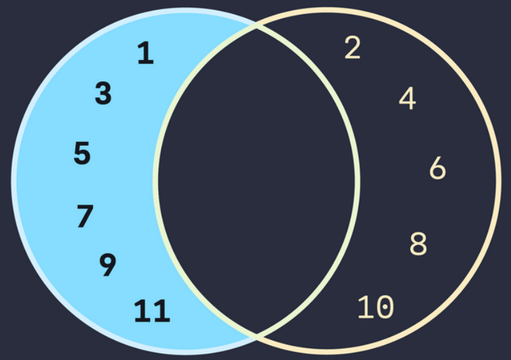
Difference

returns new set with members from left not in right

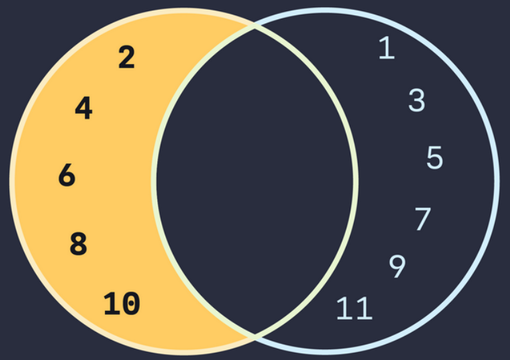
left - right




```
> set_odd - set_prime
  {1,9}
```




```
> set_odd - set_even
  {1,3,5,7,9,11}
```




```
> set_even - set_odd
  {2,4,6,8,10}
```


```
> set_odd - set_all
  {}
```

Why use sets?

- Sets make it very easy/fast to check if a value exists in a collection
- Sets are an easy way to remove duplicates from a collection