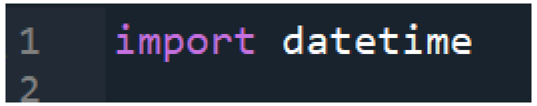
Lecture 3: Data types

Hello world!

Data types

- Numeric
  - Integers
  - Floats
- Booleans
- Strings
- List-like
  - Lists
  - Tuples
  - Sets
- Dictionaries
- Dates

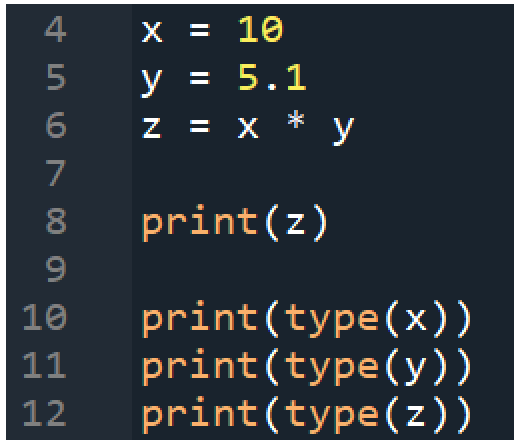
First, the imports



```python
import datetime

```

Numeric

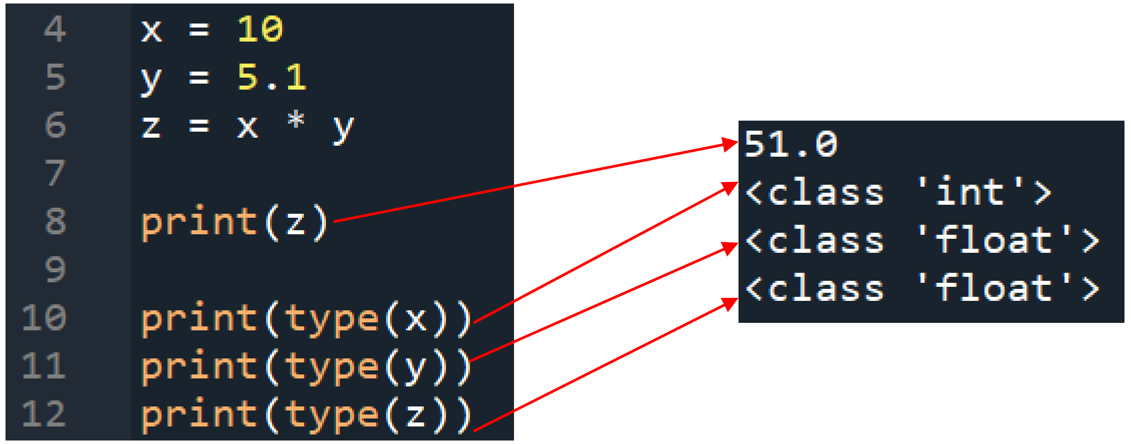


```python
x = 10
y = 5.1
z = x * y

print(z)

print(type(x))
print(type(y))
print(type(z))
```

Numeric



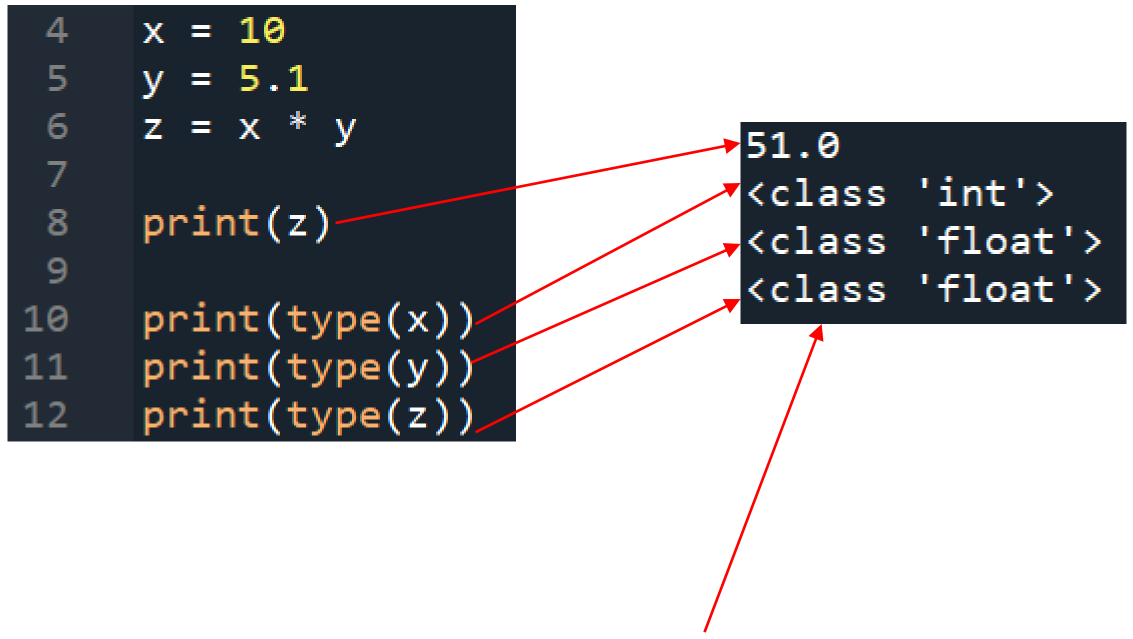
```
x = 10
y = 5.1
z = x * y

print(z)

print(type(x))
print(type(y))
print(type(z))
```

```
51.0
<class 'int'>
<class 'float'>
<class 'float'>
```

Numeric



```python
x = 10
y = 5.1
z = x * y

print(z)

print(type(x))
print(type(y))
print(type(z))
```

```
51.0
<class 'int'>
<class 'float'>
<class 'float'>
```

What does “class” mean in Python?

Interlude: classes and instances

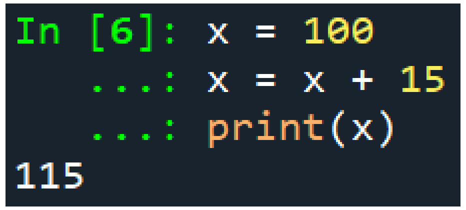
Class: the blueprint of an object

- i.e. a class isn’t the “house”, it’s the instructions for how the house will be built and what attributes it will have
- How will this object interact with special operators (e.g. + - * / ==) or with standard operations (e.g. for loops, indexing)
- What operations (functions) does this class inherently know?

Instance: the object built with the blueprint

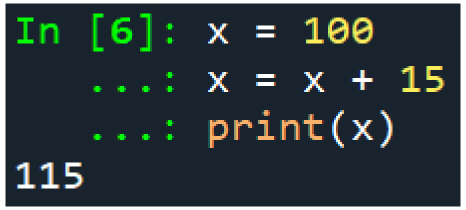
- i.e. a class tells you that a garage should go here, but you can’t park an actual car in it
- What values does a particular instance hold?

Numeric

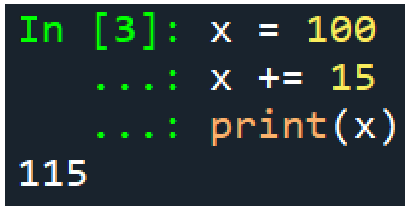


```
In [6]: x = 100
   ...: x = x + 15
   ...: print(x)
115
```

Numeric



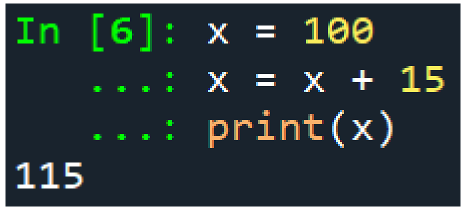
```
In [6]: x = 100
   ...: x = x + 15
   ...: print(x)
115
```



```
In [3]: x = 100
   ...: x += 15
   ...: print(x)
115
```

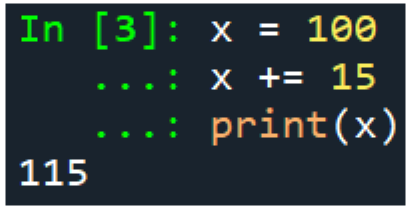
“syntactic sugar”

Numeric



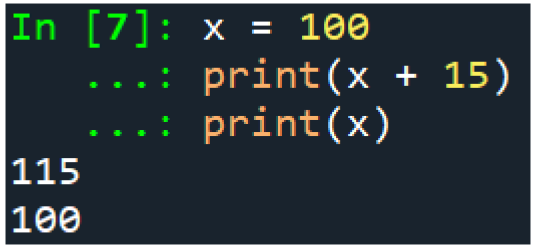
```
In [6]: x = 100
   ...: x = x + 15
   ...: print(x)
115
```

Assignment



```
In [3]: x = 100
   ...: x += 15
   ...: print(x)
115
```

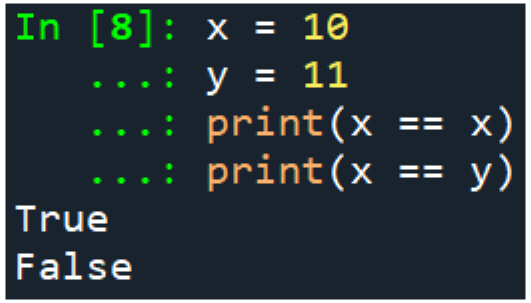
Assignment



```
In [7]: x = 100
   ...: print(x + 15)
   ...: print(x)
115
100
```

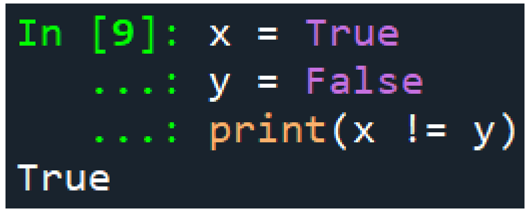
No assignment

Booleans



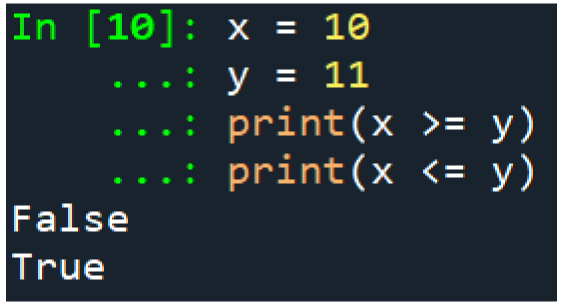
```
In [8]: x = 10
   ...: y = 11
   ...: print(x == x)
   ...: print(x == y)
True
False
```

Booleans



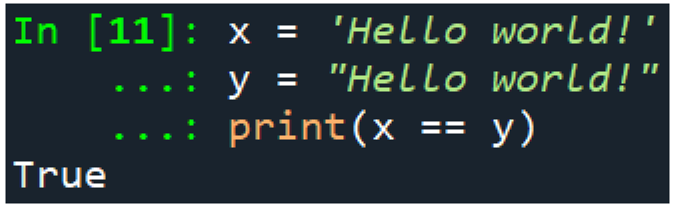
```
In [9]: x = True
   ...: y = False
   ...: print(x != y)
True
```

Booleans



```
In [10]: x = 10
    ...: y = 11
    ...: print(x >= y)
    ...: print(x <= y)
False
True
```

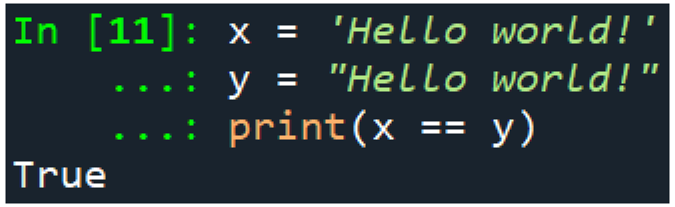
Strings



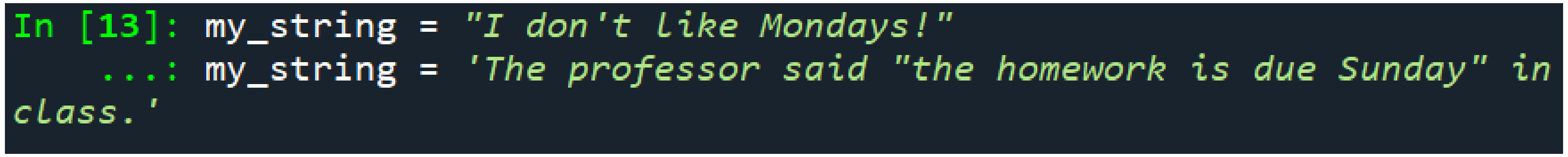
```
In [11]: x = 'Hello world!'
    ...: y = "Hello world!"
    ...: print(x == y)
True
```

Single quotes
Double quotes
Tests of equality

Strings

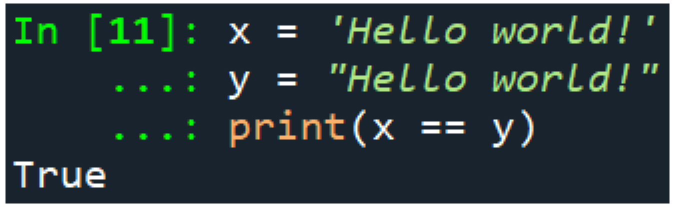


```
In [11]: x = 'Hello world!'
    ...: y = "Hello world!"
    ...: print(x == y)
True
```

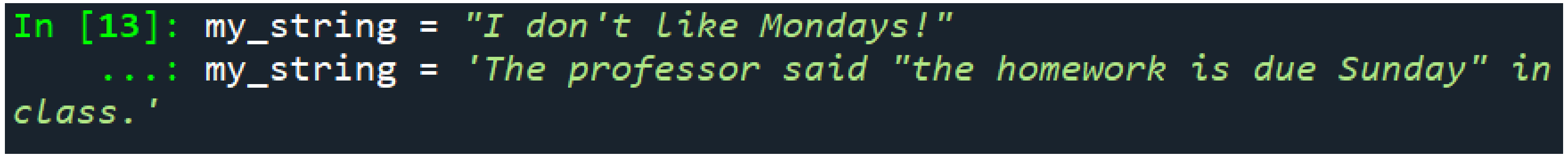


```
In [13]: my_string = "I don't like Mondays!"
    ...: my_string = 'The professor said "the homework is due Sunday" in
class.'
```

Strings



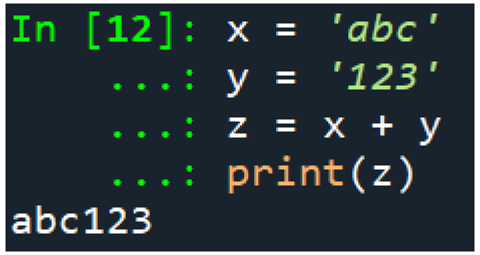
```
In [11]: x = 'Hello world!'
    ...: y = "Hello world!"
    ...: print(x == y)
True
```



```
In [13]: my_string = "I don't like Mondays!"
    ...: my_string = 'The professor said "the homework is due Sunday" in
class.'
```

Otherwise, pick what you like and then BE CONSISTENT!

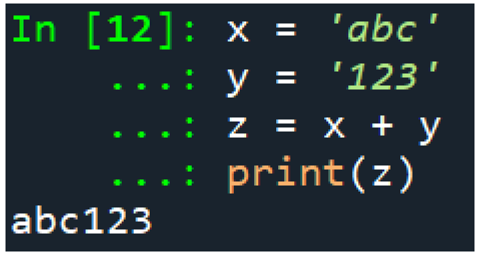
Strings



```
In [12]: x = 'abc'
    ...: y = '123'
    ...: z = x + y
    ...: print(z)
abc123
```

Easy concatenation with + operator

Strings

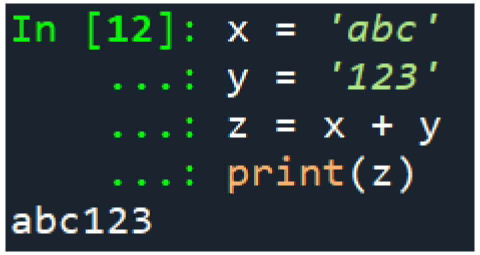


```
In [12]: x = 'abc'
    ...: y = '123'
    ...: z = x + y
    ...: print(z)
abc123
```

Easy concatenation with + operator

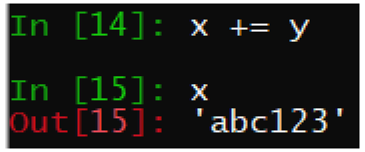
Hey, does that mean we can use += with strings?

Strings



```
In [12]: x = 'abc'
    ...: y = '123'
    ...: z = x + y
    ...: print(z)
abc123
```

Easy concatenation with + operator

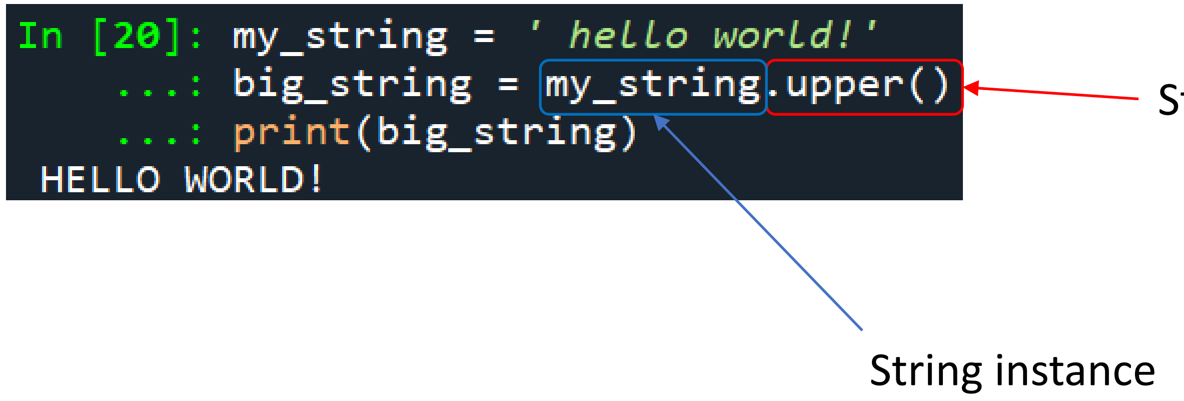


```
In [14]: x += y

In [15]: x
Out[15]: 'abc123'
```

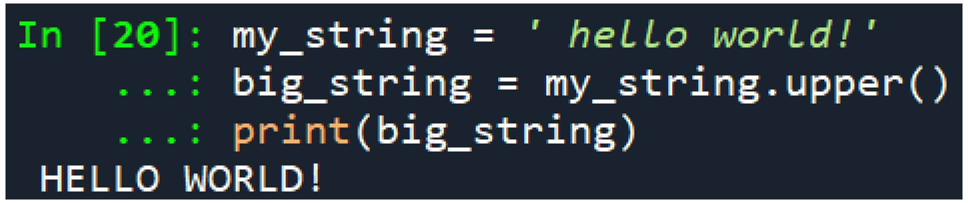
Why yes we can!

String “methods”

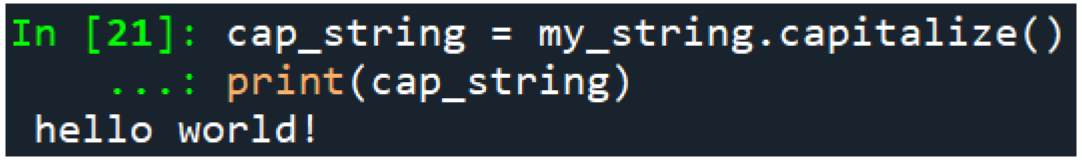


```
In [20]: my_string = ' hello world!'
    ...: big_string = my_string.upper()
    ...: print(big_string)
 HELLO WORLD!
```

String “methods”

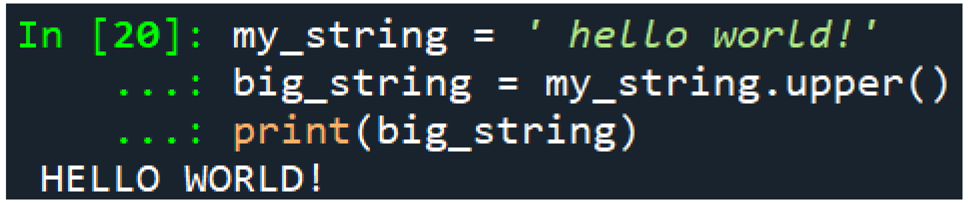


```
In [20]: my_string = ' hello world!'
    ...: big_string = my_string.upper()
    ...: print(big_string)
 HELLO WORLD!
```

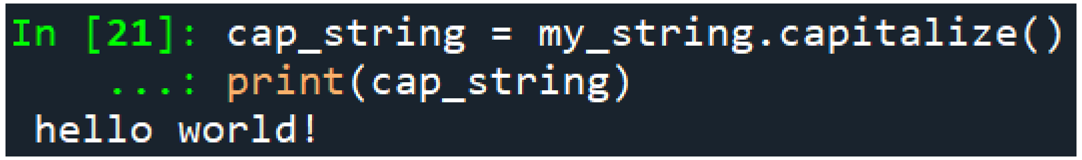


```
In [21]: cap_string = my_string.capitalize()
    ...: print(cap_string)
 hello world!
```

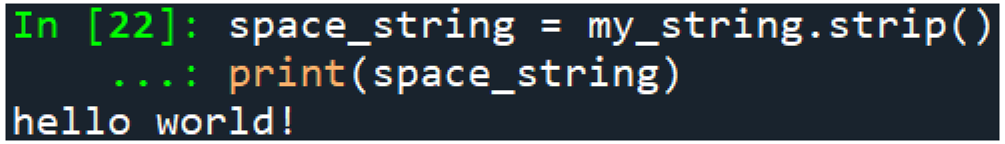
String “methods”



```
In [20]: my_string = ' hello world!'
    ...: big_string = my_string.upper()
    ...: print(big_string)
 HELLO WORLD!
```

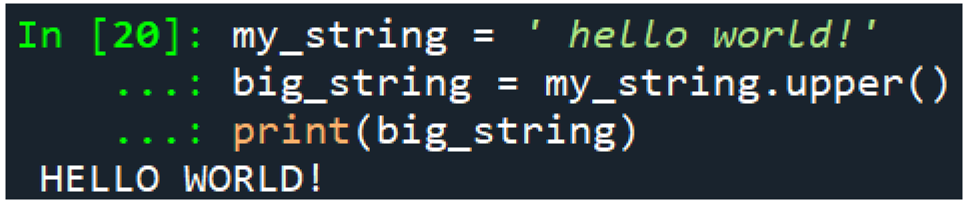


```
In [21]: cap_string = my_string.capitalize()
    ...: print(cap_string)
 hello world!
```

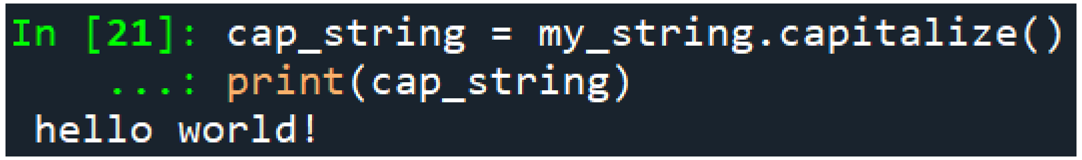


```
In [22]: space_string = my_string.strip()
    ...: print(space_string)
hello world!
```

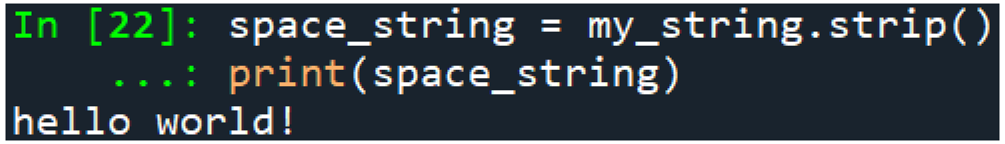
String “methods”



```
In [20]: my_string = ' hello world!'
    ...: big_string = my_string.upper()
    ...: print(big_string)
 HELLO WORLD!
```

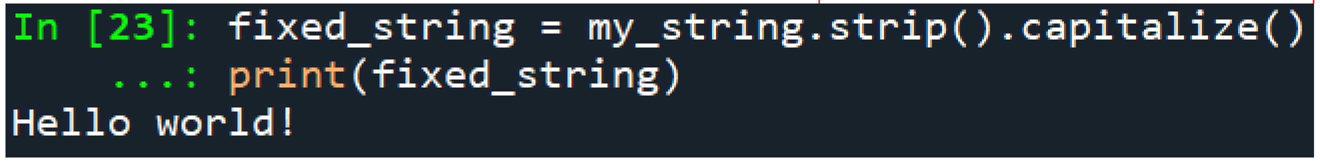


```
In [21]: cap_string = my_string.capitalize()
    ...: print(cap_string)
 hello world!
```



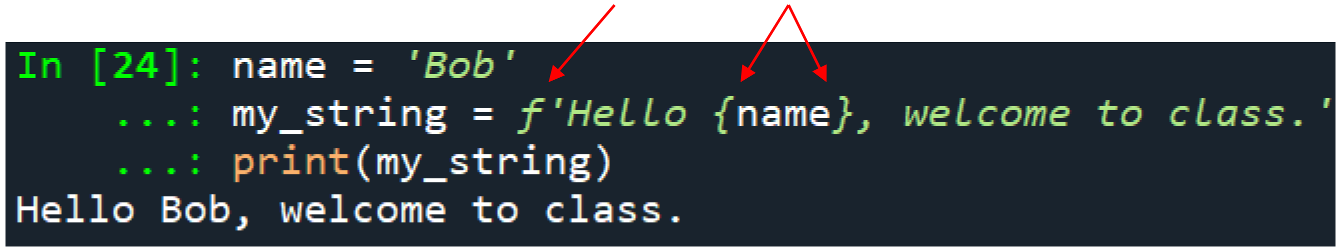
```
In [22]: space_string = my_string.strip()
    ...: print(space_string)
hello world!
```

We can call methods one after the other



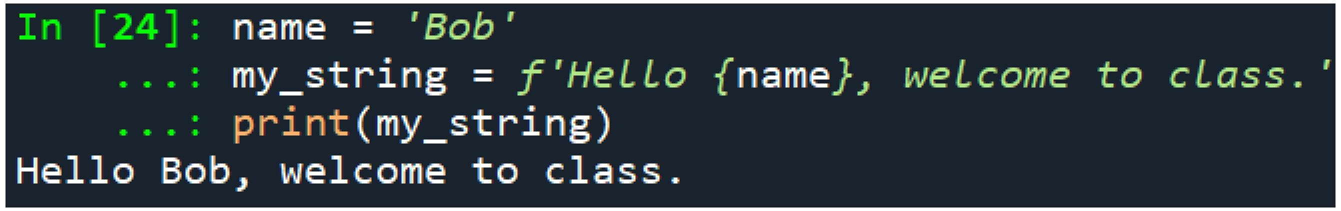
```
In [23]: fixed_string = my_string.strip().capitalize()
    ...: print(fixed_string)
Hello world!
```

String formatting

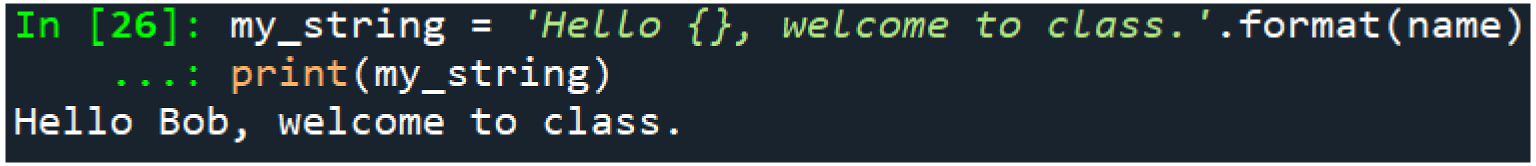


```
In [24]: name = 'Bob'
    ...: my_string = f'Hello {name}, welcome to class.'
    ...: print(my_string)
Hello Bob, welcome to class.
```

String formatting

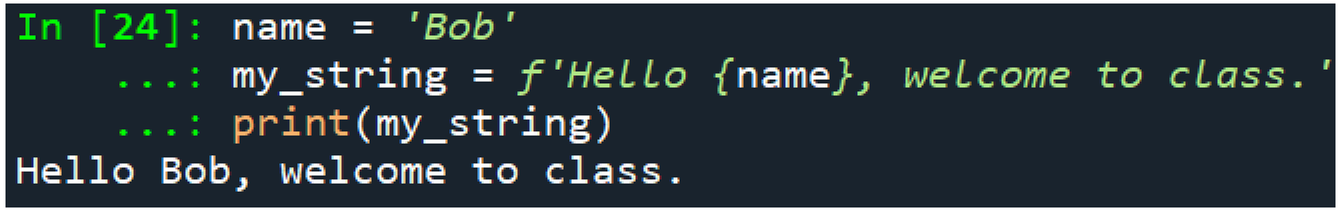


```
In [24]: name = 'Bob'
    ...: my_string = f'Hello {name}, welcome to class.'
    ...: print(my_string)
Hello Bob, welcome to class.
```

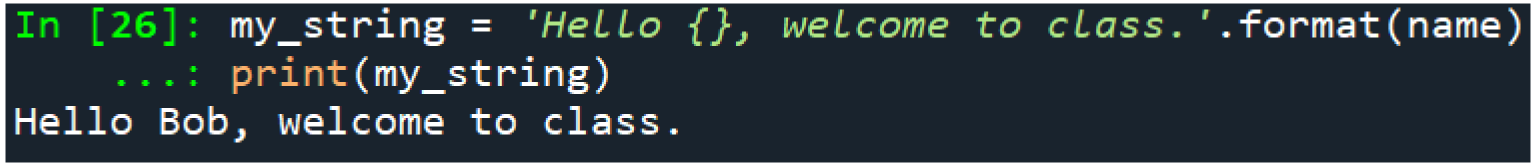


```
In [26]: my_string = 'Hello {}, welcome to class.'.format(name)
    ...: print(my_string)
Hello Bob, welcome to class.
```

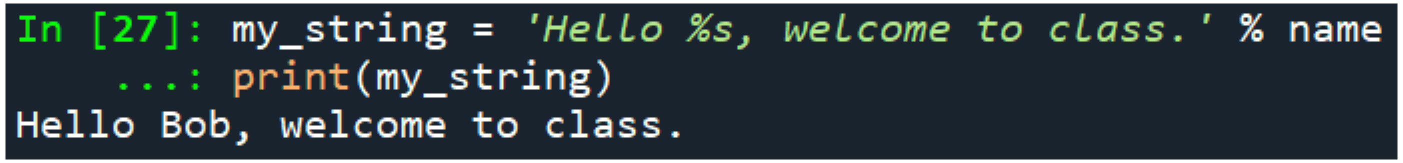
String formatting



```
In [24]: name = 'Bob'
    ...: my_string = f'Hello {name}, welcome to class.'
    ...: print(my_string)
Hello Bob, welcome to class.
```

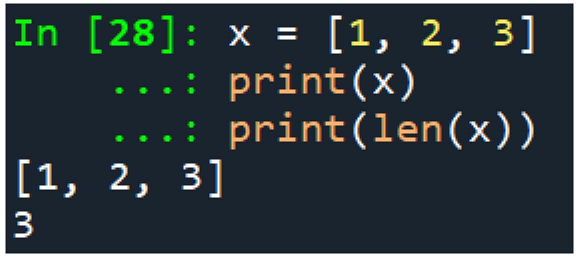


```
In [26]: my_string = 'Hello {}, welcome to class.'.format(name)
    ...: print(my_string)
Hello Bob, welcome to class.
```



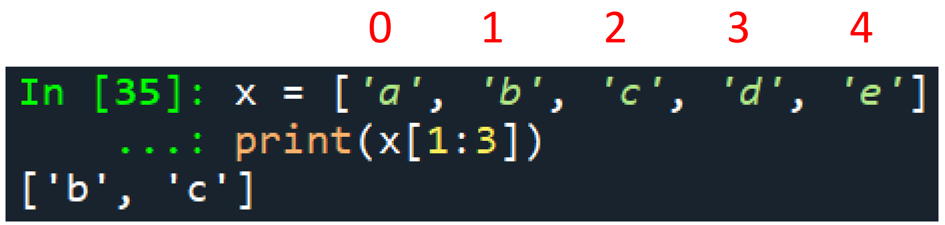
```
In [27]: my_string = 'Hello %s, welcome to class.' % name
    ...: print(my_string)
Hello Bob, welcome to class.
```

Lists



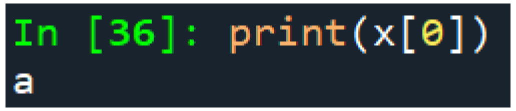
```
In [28]: x = [1, 2, 3]
    ...: print(x)
    ...: print(len(x))
[1, 2, 3]
3
```

Lists



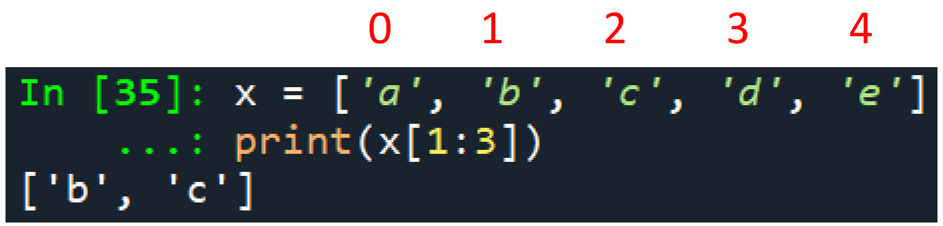
```
In [35]: x = ['a', 'b', 'c', 'd', 'e']
    ...: print(x[1:3])
['b', 'c']
```

my_list[start:stop:step]



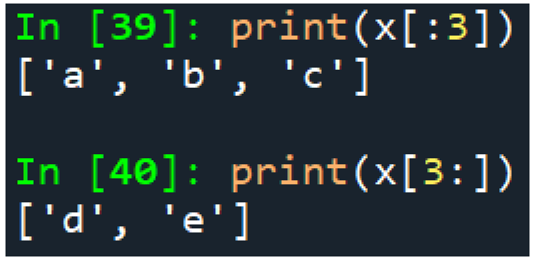
```
In [36]: print(x[0])
a
```

Lists



```
In [35]: x = ['a', 'b', 'c', 'd', 'e']
    ...: print(x[1:3])
['b', 'c']
```

my_list[start:stop:step]



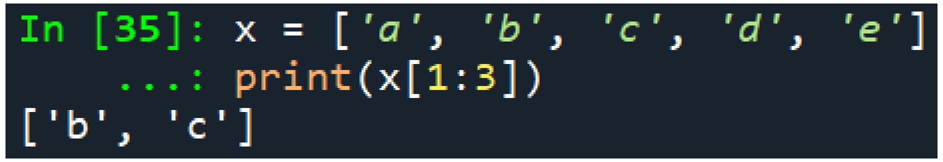
```
In [39]: print(x[:3])
['a', 'b', 'c']

In [40]: print(x[3:])
['d', 'e']
```

Why closed on the left, open on the right?

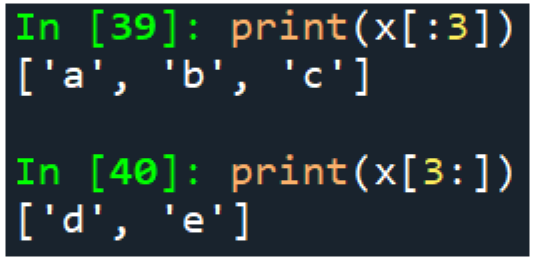
Lists

0 1 2 3 4



```
In [35]: x = ['a', 'b', 'c', 'd', 'e']
    ...: print(x[1:3])
['b', 'c']
```

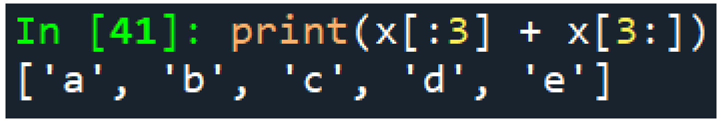
my_list[start:stop:step]



```
In [39]: print(x[:3])
['a', 'b', 'c']

In [40]: print(x[3:])
['d', 'e']
```

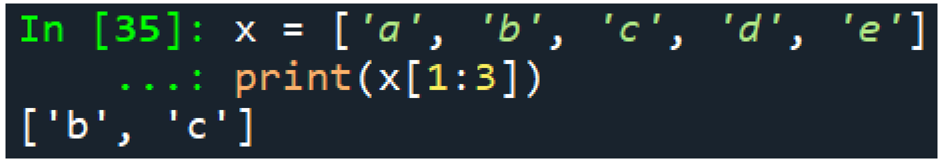
Why closed on the left, open on the right?



```
In [41]: print(x[:3] + x[3:])
['a', 'b', 'c', 'd', 'e']
```

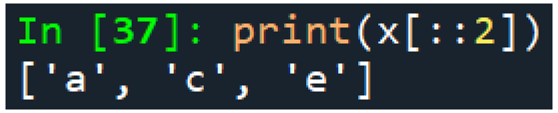
Lists

0 1 2 3 4



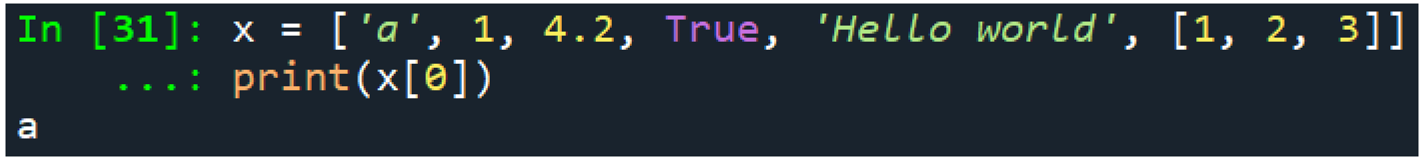
```
In [35]: x = ['a', 'b', 'c', 'd', 'e']
    ...: print(x[1:3])
['b', 'c']
```

my_list[start:stop:step]



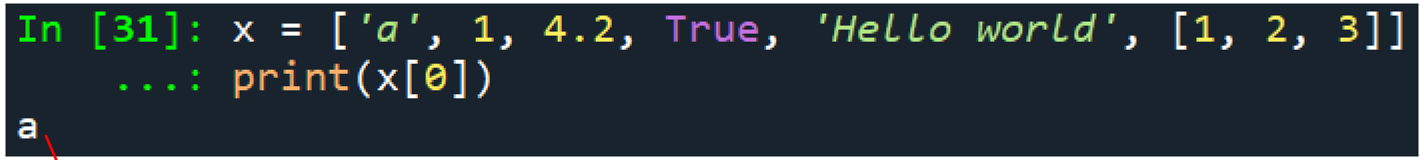
```
In [37]: print(x[::2])
['a', 'c', 'e']
```

Lists

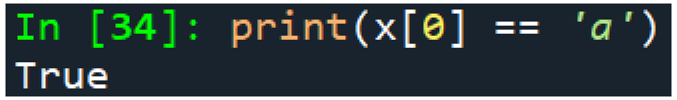


```
In [31]: x = ['a', 1, 4.2, True, 'Hello world', [1, 2, 3]]
    ...: print(x[0])
a
```

Lists

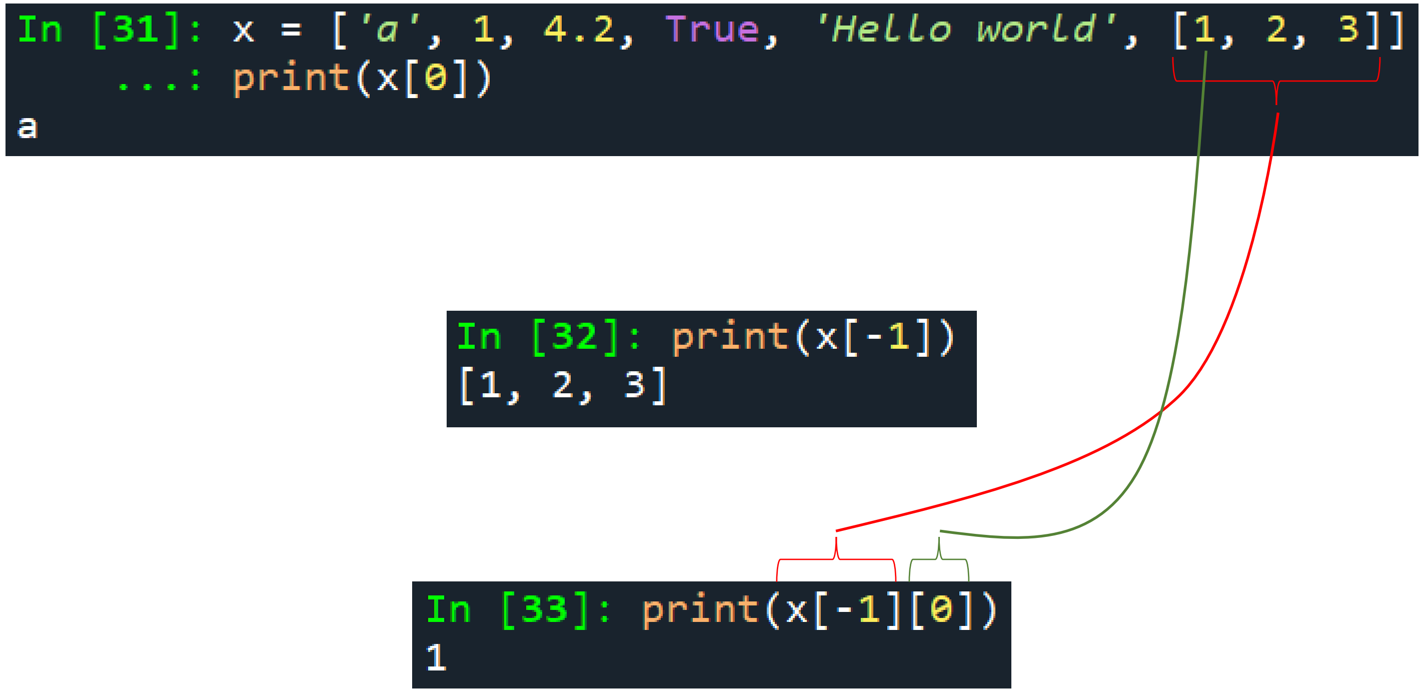


```
In [31]: x = ['a', 1, 4.2, True, 'Hello world', [1, 2, 3]]
    ...: print(x[0])
a
```



```
In [34]: print(x[0] == 'a')
True
```

Lists

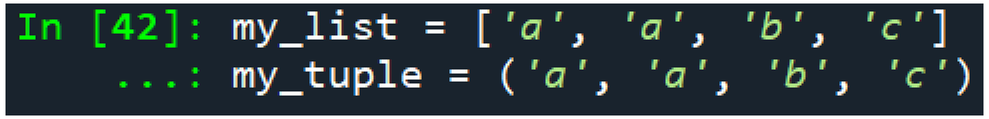


```
In [31]: x = ['a', 1, 4.2, True, 'Hello world', [1, 2, 3]]
    ...: print(x[0])
a
```

```
In [32]: print(x[-1])
[1, 2, 3]
```

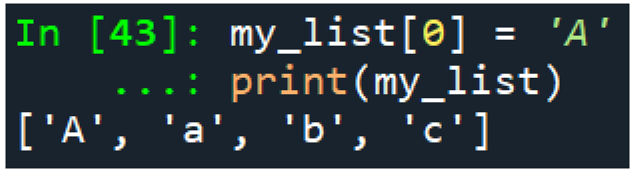
```
In [33]: print(x[-1][0])
1
```

Other list-like objects



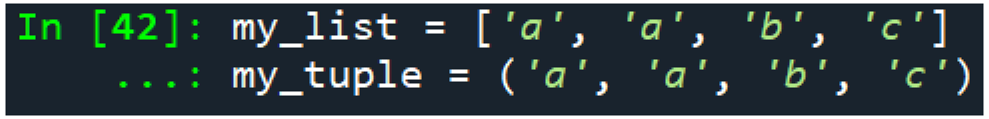
```
In [42]: my_list = ['a', 'a', 'b', 'c']
    ...: my_tuple = ('a', 'a', 'b', 'c')
```

Assignment in a list



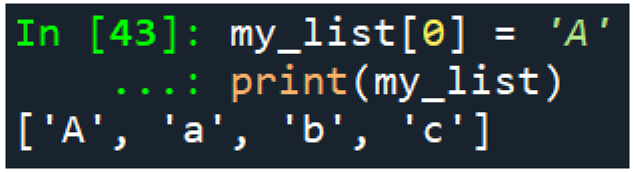
```
In [43]: my_list[0] = 'A'
    ...: print(my_list)
['A', 'a', 'b', 'c']
```

Other list-like objects



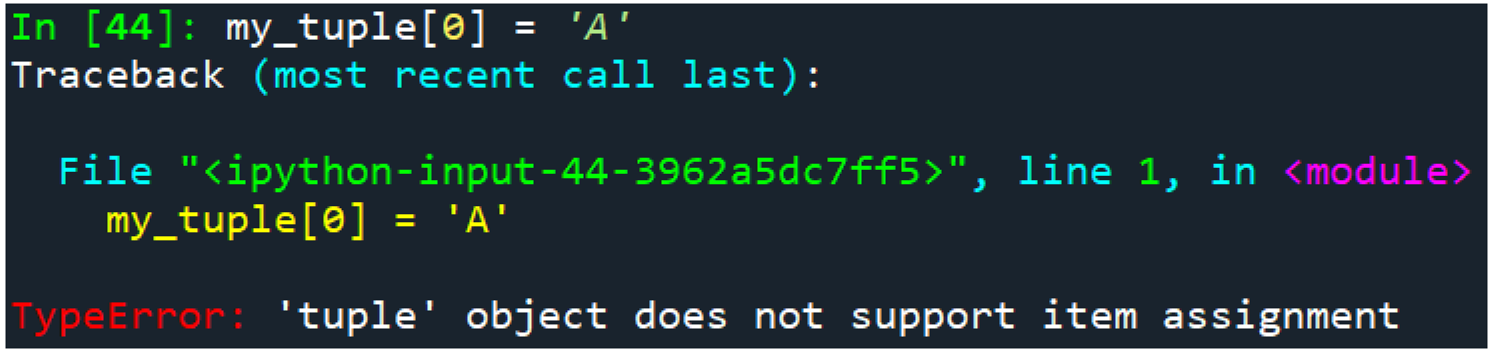
```
In [42]: my_list = ['a', 'a', 'b', 'c']
    ...: my_tuple = ('a', 'a', 'b', 'c')
```

Assignment in a list



```
In [43]: my_list[0] = 'A'
    ...: print(my_list)
['A', 'a', 'b', 'c']
```

Assignment in a tuple

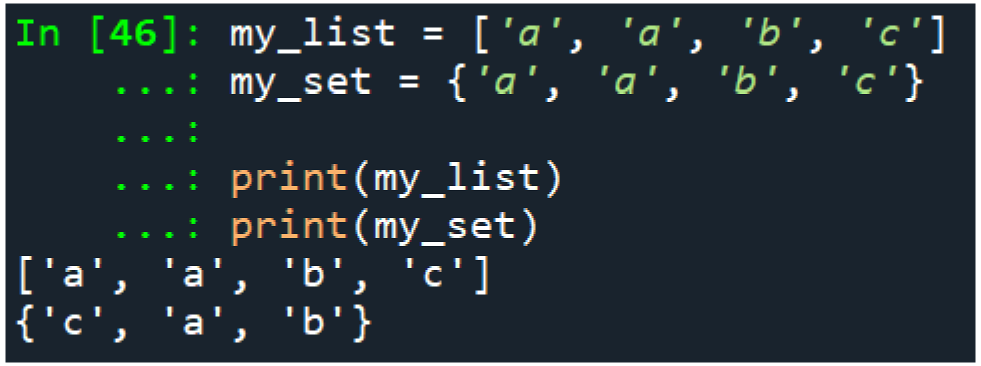


```
In [44]: my_tuple[0] = 'A'
Traceback (most recent call last):

  File "<ipython-input-44-3962a5dc7ff5>", line 1, in <module>
    my_tuple[0] = 'A'

TypeError: 'tuple' object does not support item assignment
```

Other list-like objects

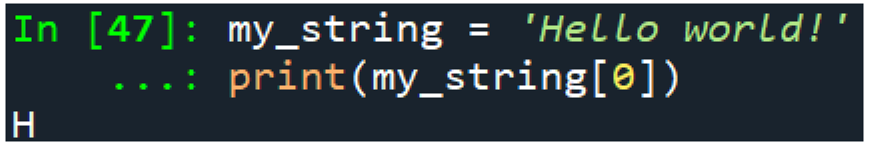


```
In [46]: my_list = ['a', 'a', 'b', 'c']
    ...: my_set = {'a', 'a', 'b', 'c'}
    ...:
    ...: print(my_list)
    ...: print(my_set)
['a', 'a', 'b', 'c']
{'c', 'a', 'b'}
```

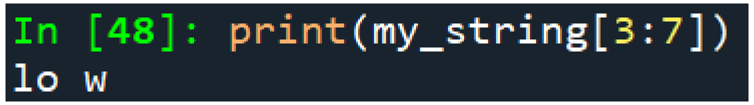
Sets enforce unique values

Sets do not maintain ordering

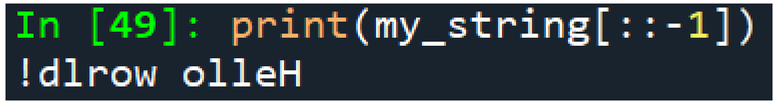
Other list-like objects



```
In [47]: my_string = 'Hello world!'
    ...: print(my_string[0])
H
```

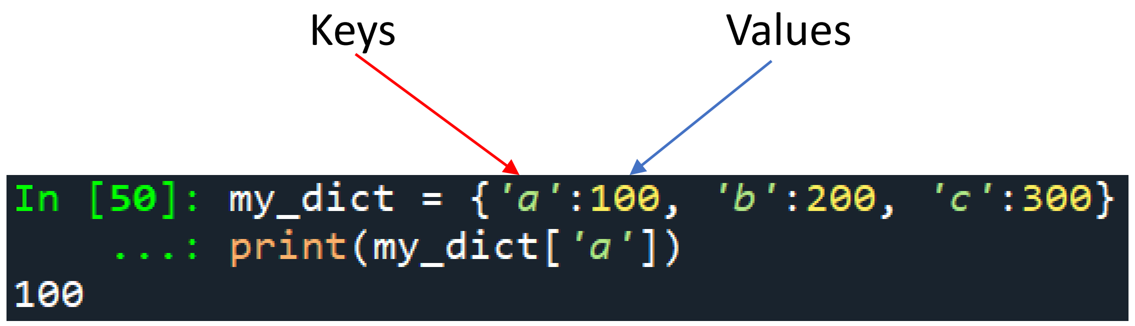


```
In [48]: print(my_string[3:7])
lo w
```



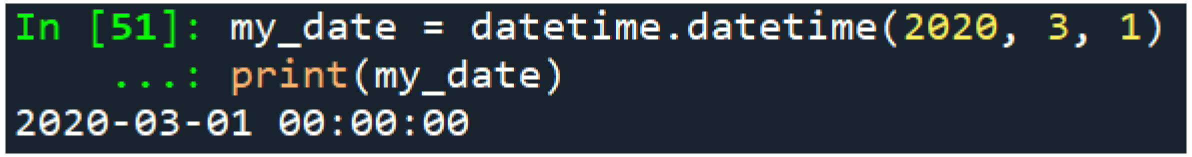
```
In [49]: print(my_string[::-1])
!dlrow olleH
```

Dictionaries



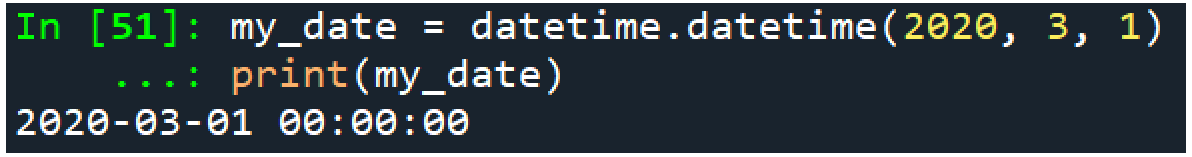
```
In [50]: my_dict = {'a':100, 'b':200, 'c':300}
    ...: print(my_dict['a'])
100
```

Dates

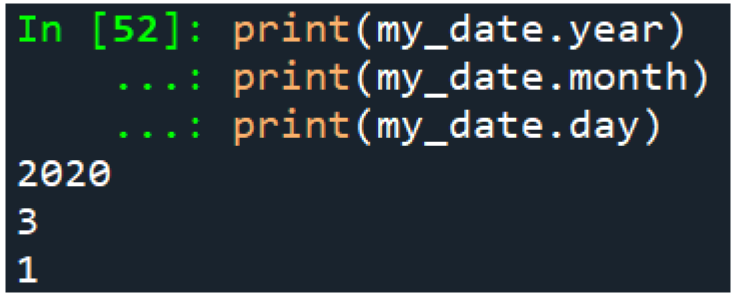


```
In [51]: my_date = datetime.datetime(2020, 3, 1)
    ...: print(my_date)
2020-03-01 00:00:00
```

Dates

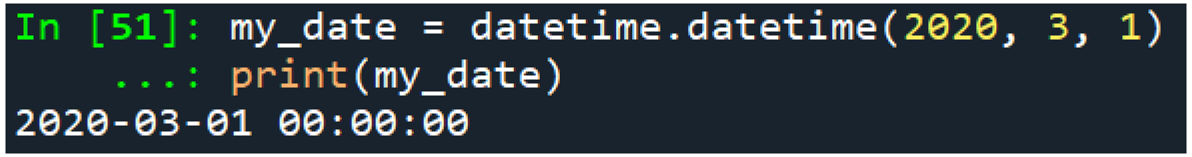


```
In [51]: my_date = datetime.datetime(2020, 3, 1)
    ...: print(my_date)
2020-03-01 00:00:00
```

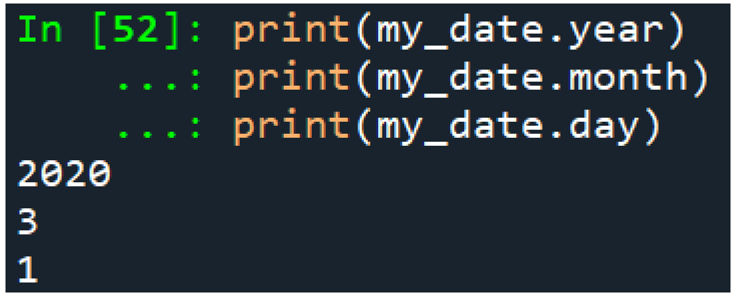


```
In [52]: print(my_date.year)
    ...: print(my_date.month)
    ...: print(my_date.day)
2020
3
1
```

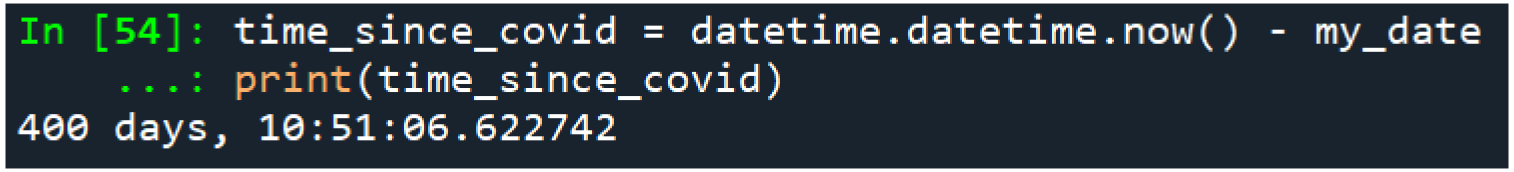
Dates



```
In [51]: my_date = datetime.datetime(2020, 3, 1)
    ...: print(my_date)
2020-03-01 00:00:00
```



```
In [52]: print(my_date.year)
    ...: print(my_date.month)
    ...: print(my_date.day)
2020
3
1
```



```
In [54]: time_since_covid = datetime.datetime.now() - my_date
    ...: print(time_since_covid)
400 days, 10:51:06.622742
```