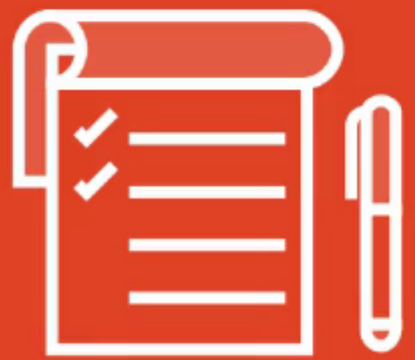
Overview



`str`, `bytes`, `list`, and `dict`

`for`-loop

Put it all together

Collections

Collections

`str`

`bytes`

`list`

`dict`

`for-loops`

str

Data type for strings in Python

Sequence of Unicode code points

Immutable

String Literals

```
>>> 'This is a string'
'This is a string'
>>> "This is also a string"
'This is also a string'
>>> "inconsistent'
  File "<stdin>", line 1
    "inconsistent'
                  ^
SyntaxError: EOL while scanning string literal
>>> "It's a good thing."
"It's a good thing."
>>> '"Yes!", he said, "I agree!"'
'"Yes!", he said, "I agree!"'
>>>
```

String Literals

```
>>> "first" "second"
'firstsecond'
>>>
```

Strings with Newlines

Multiline strings

Spread the literal across multiple lines

Escape sequences

Embed escape sequences in a single-line literal

Multiline Strings

```
>>> """This is
... a multiline
... string"""
'This is\na multiline\nstring'
>>> '''So
... is
... this.'''
'So\nis\nthis.'
>>> m = 'This string\nspans multiple\nlines'
>>> m
'This string\nspans multiple\nlines'
>>> print(m)
This string
spans multiple
lines
>>>
```

Newlines and Operating Systems

Windows Carriage-return, line-feed

\r\n

Linux and macOS Carriage-return

\r

Universal Newlines

Python translates `\n` to the appropriate newline sequence for your platform

PEP 278: python.org/dev/peps/pep-0278/

Escape Sequences

```
>>> "This is a \" in a string"
'This is a " in a string'
>>> 'This is a \' in a string'
"This is a ' in a string"
>>> 'This is a \" and a \' in a string'
'This is a " and a \' in a string'
>>> k = 'A \\ in a string'
>>> k
'A \\ in a string'
>>> print(k)
A \ in a string
>>>
```

All Escape Sequences

Sequence	Meaning
\newline	Backslash and newline ignored
\\	Backslash (\)
\'	Single quotes (')
\"	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	Character with octal value *ooo*
\xhh	Character with hex value *hh*
Only recognized in string literals	

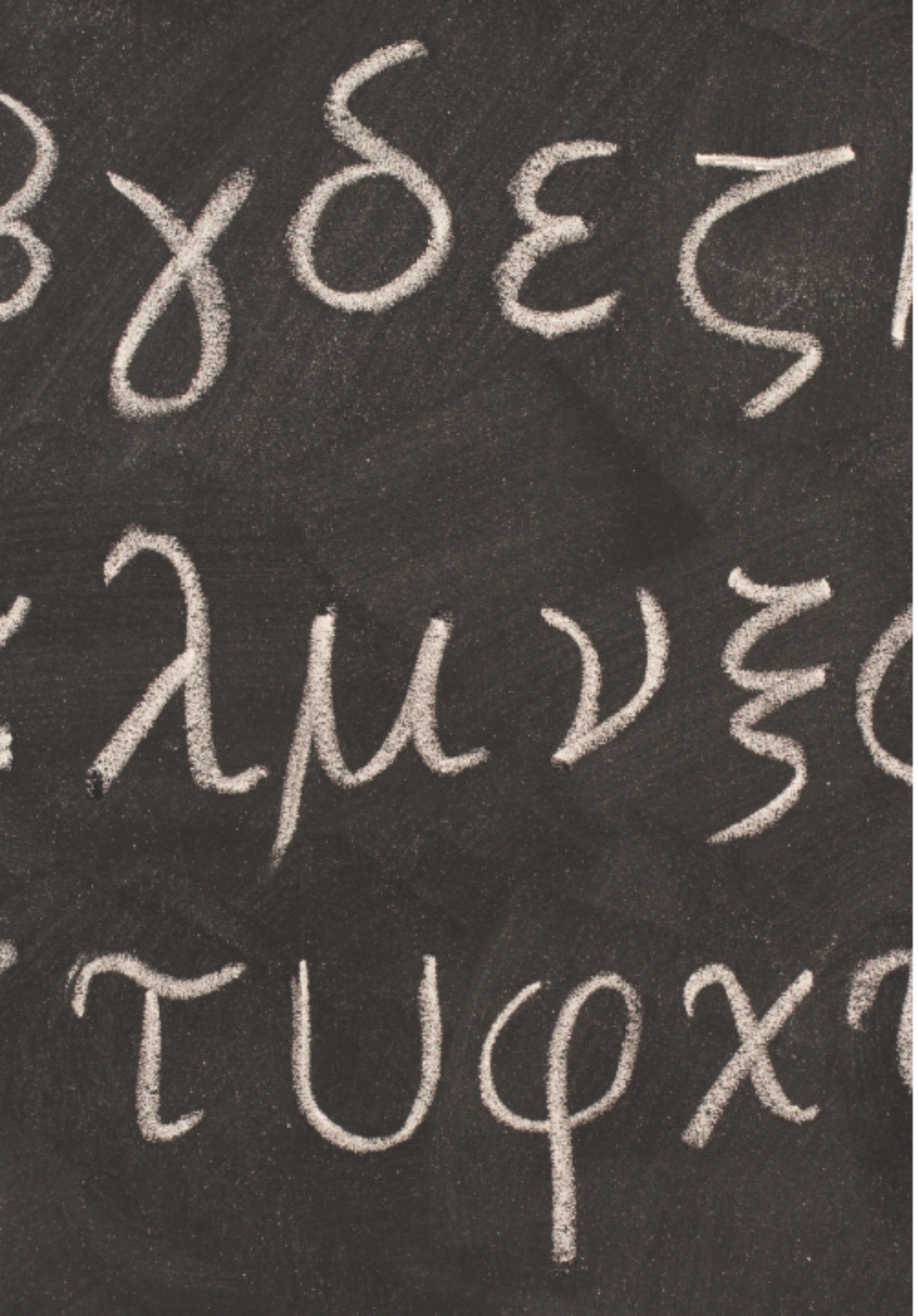
String Features

```
>>> path = r'C:\Users\Merlin\Documents\Spells'
>>> path
'C:\\Users\\Merlin\\Documents\\Spells'
>>> print(path)
C:\Users\Merlin\Documents\Spells
>>> str(496)
'496'
>>> str(6.02e23)
'6.02e+23'
>>> s = 'parrot'
>>> s[4]
'o'
>>> type(s[4])
<class 'str'>
>>>
```

String Features

```
 |
 |  __sizeof__(self, /)
 |      Return the size of the string in memory, in bytes.
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  capitalize(self, /)
 |      Return a capitalized version of the string.
 |
 |      More specifically, make the first character have upper case and the rest
lower
 |      case.
 |
 |  casefold(self, /)
 |      Return a version of the string suitable for caseless comparisons.
 |
```

```
>>> c = "oslo"
>>> c.capitalize()
'Oslo'
>>> c
'oslo'
>>>
```



`str` is Unicode

Python 3 source encoding is UTF-8

Unicode in Strings

```
>>> "Vi er så glad for å høre og lære om Python!"
'Vi er så glad for å høre og lære om Python!'
>>> "Vi er s\u00e5 glad for \u00e5 h\xf8re og l\u00e6re om Python!"
'Vi er så glad for å høre og lære om Python!'
>>> '\xe5'
'å'
>>> '\345'
'å'
>>>
```

bytes

Data type for sequences of bytes

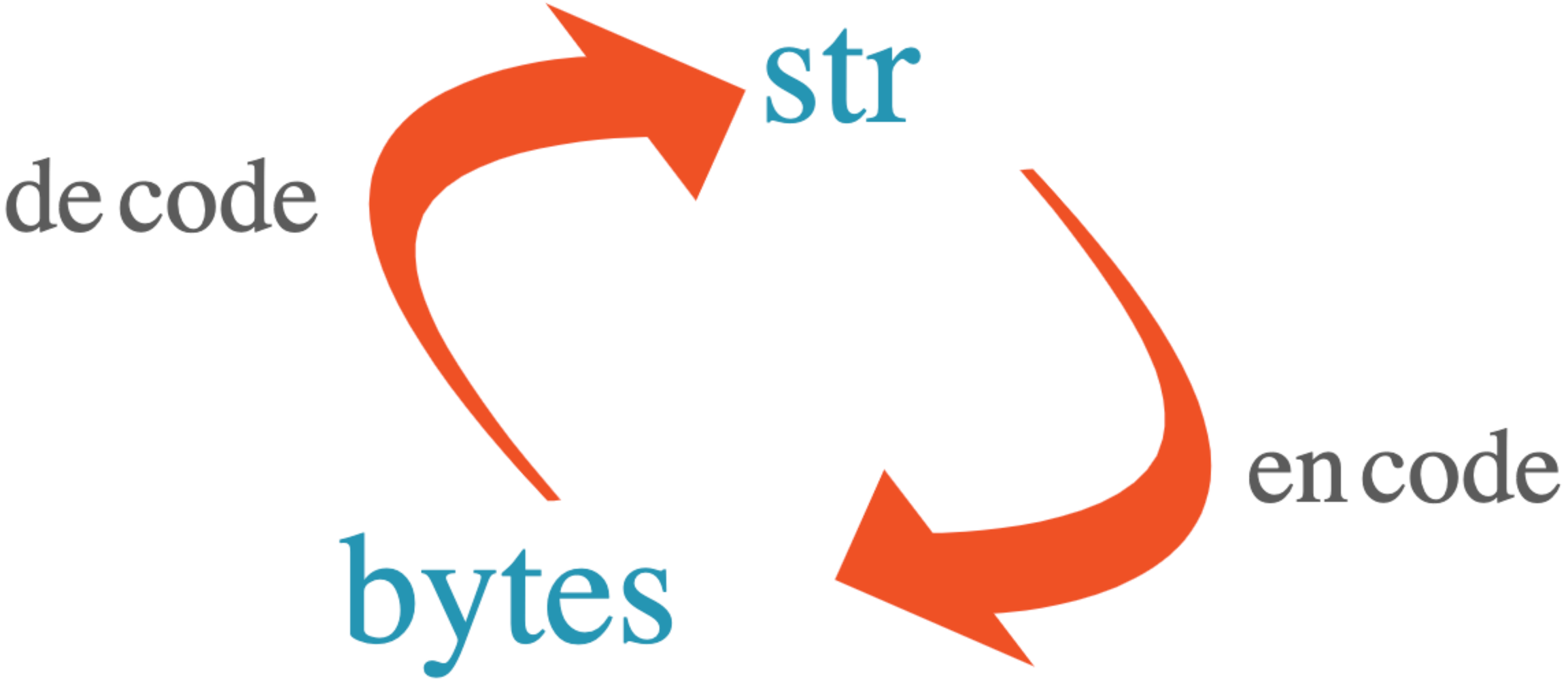
Raw binary data

Fixed-width single-byte encodings

Bytes

```
>>> b'data'
b'data'
>>> b"data"
b'data'
>>> d = b'some bytes'
>>> d[0]
115
>>> d.split()
[b'some', b'bytes']
>>>
```

Converting Betweeen Strings and Bytes



docs.python.org/3/library/codecs.html#standard-encodings

Encode/Decode

```
>>> norsk = "Jeg begynte å fortære en sandwich mens jeg kjørte taxi på vei til quiz"
>>> data = norsk.encode('utf8')
>>> data
b'Jeg begynte \xc3\xa5 fort\xc3\xa6re en sandwich mens jeg kj\xc3\xb8rte taxi p\xc3\
xa5 vei til quiz'
>>> norwegian = data.decode('utf8')
>>> norwegian == norsk
True
>>> norwegian
'Jeg begynte å fortære en sandwich mens jeg kjørte taxi på vei til quiz'
>>>
```

list

Sequences of objects

Mutable

A workhorse in Python

Lists

```
>>> [1, 9, 8]
[1, 9, 8]
>>> a = ["apple", "orange", "pear"]
>>> a[1]
'orange'
>>> a[1] = 7
>>> a
['apple', 7, 'pear']
>>> b = []
>>> b.append(1.618)
>>> b
[1.618]
>>> b.append(1.414)
>>> b
[1.618, 1.414]
>>> list("characters")
['c', 'h', 'a', 'r', 'a', 'c', 't', 'e', 'r', 's']
>>> c = ['bear',
...      'giraffe',
...      'elephant',
...      'caterpillar',]
>>> c
['bear', 'giraffe', 'elephant', 'caterpillar']
>>>
```

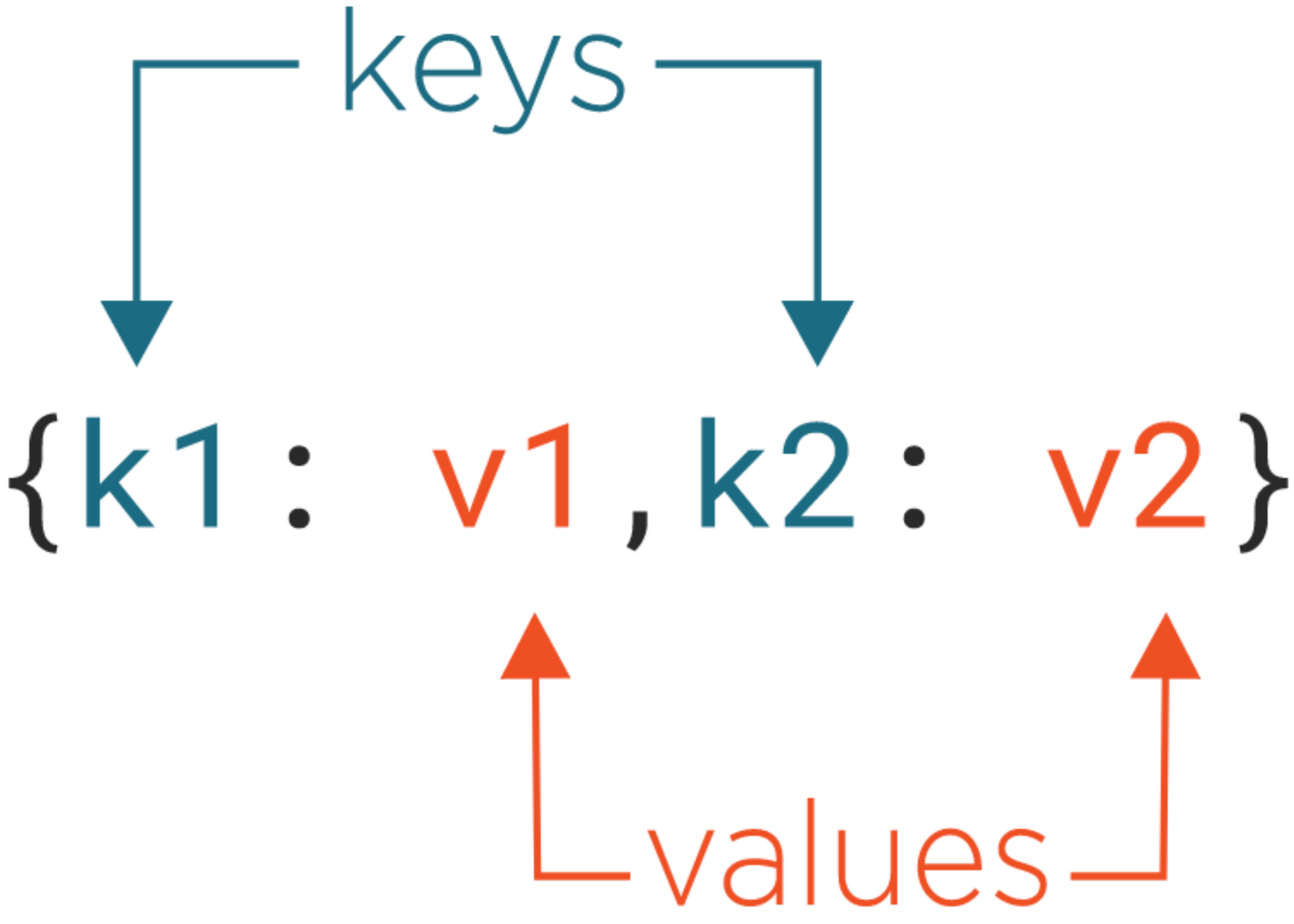
dict

Fundamental data structure in Python

Map keys to values

Also known as maps or associative arrays

Dict Literals



Dict

```
>>> d = {'alice': '878-8728-922', 'bob': '256-5262-124', 'eve': '198-2321-787'}
>>> d['alice']
'878-8728-922'
>>> d['alice'] = '966-4532-6272'
>>> d
{'alice': '966-4532-6272', 'bob': '256-5262-124', 'eve': '198-2321-787'}
>>> d['charles'] = '334-5551-913'
>>> d
{'alice': '966-4532-6272', 'bob': '256-5262-124', 'eve': '198-2321-787', 'charle
s': '334-5551-913'}
>>> e = {}
>>>
```

for-loop

Visit each item in an iterable sequence

For-loops

```
for item in iterable:
    ...body...
```

For-loop

```
>>> cities = ["London", "New York", "Paris", "Oslo", "Helsinki"]
>>> for city in cities:
...     print(city)
...
London
New York
Paris
Oslo
Helsinki
>>> colors = {'crimson': 0xdc143c, 'coral': 0xff7f50, 'teal': 0x008080}
>>> for color in colors:
...     print(color, colors[color])
...
crimson 14423100
coral 16744272
teal 32896
>>>
```

Putting It All Together

```
or', b'good', b'or', b'for', b'evil', b'in', b'the', b'superlative', b'degree',
b'of', b'comparison', b'only']
>>>
>>>
>>>
.decode('utf8').split()
...
...
...
>>>
>>>
['It', 'was', 'the', 'best', 'of', 'times', 'it', 'was', 'the', 'worst', 'of', '
times', 'it', 'was', 'the', 'age', 'of', 'wisdom', 'it', 'was', 'the', 'age', 'o
f', 'foolishness', 'it', 'was', 'the', 'epoch', 'of', 'belief', 'it', 'was', 'th
e', 'epoch', 'of', 'incredulity', 'it', 'was', 'the', 'season', 'of', 'Light', '
it', 'was', 'the', 'season', 'of', 'Darkness', 'it', 'was', 'the', 'spring', 'of
', 'hope', 'it', 'was', 'the', 'winter', 'of', 'despair', 'we', 'had', 'everythi
ng', 'before', 'us', 'we', 'had', 'nothing', 'before', 'us', 'we', 'were', 'all'
, 'going', 'direct', 'to', 'Heaven', 'we', 'were', 'all', 'going', 'direct', 'th
e', 'other', 'way', 'in', 'short', 'the', 'period', 'was', 'so', 'far', 'like',
'the', 'present', 'period', 'that', 'some', 'of', 'its', 'noisiest', 'authoritie
s', 'insisted', 'on', 'its', 'being', 'received', 'for', 'good', 'or', 'for', 'e
vil', 'in', 'the', 'superlative', 'degree', 'of', 'comparison', 'only']
>>>
```

Check for Errors

Carefully check and re-enter code if there are errors.

An `HTTPError` indicates a network problem.

Recall Bytes

Bytes literals prefixed with lowercase 'b'

HTTP data is provided as bytes

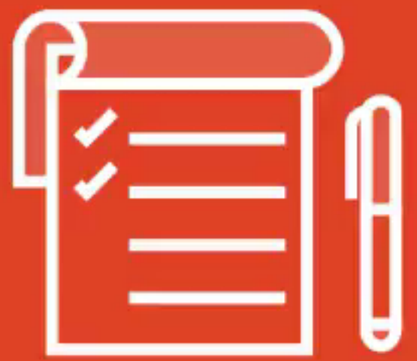
Use `bytes.decode()` to get strings

From REPL to IDE

The REPL is good for short-lived work and experimentation.

Use an editor/IDE for larger or longer-lived projects.

Summary



Strings

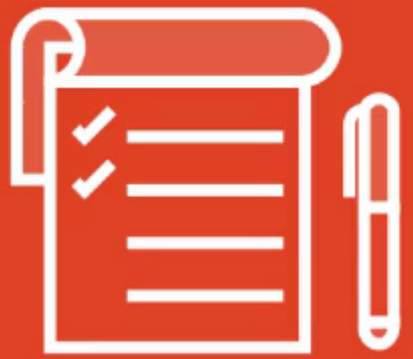
- Single- and multi-line literals
- Concatenation of adjacent literals
- Universal newlines
- Escape sequences
- Raw strings
- Use `str` constructor to convert other types
- Access individual characters with square bracket indexing
- Rich API
- String literals can contain Unicode

Summary

Bytes

- Sequence of bytes rather than codepoints
- Literals prefixed with lowercase "b"
- Use `str.encode()` and `bytes.decode()` for conversion

Summary



Lists

- Mutable, heterogeneous sequences
- Literals delimited by square brackets
- Literal items separated by commas
- Access elements with square brackets
- Elements can be replaced by assigning to an index
- Grow lists with `append()`
- Use `list` constructor to create lists from other sequences

Summary

Dicts

- Associate keys with values
- Literals are delimited by curly braces
- Key-value pairs are separated by commas
- Keys are separated from values by colons

For-loops

- Bind each item from an iterable one at a time to a name
- Called for-each loops in other languages