

Strings and Representations

Austin Bingham
🐦 @austin_bingham
austin@sixty-north.com



Presenter

Robert Smallshire
🐦 @robsmallshire
rob@sixty-north.com



pluralsight 
hardcore dev and IT training



`str()` `repr()`

functions for making
string representations
from Python objects
which call the methods

`__str__()`

`__repr__()`

The **built-in** function

repr()

produces an **unambiguous** string
representation of an object.

```
Point2D(x=42, y=69)
```



repr()

- **Exactness is more important than human-friendliness**
- **Suited for debugging**
- **Includes identifying information**
- **Generally best for logging**



`repr()` is for *developers*

The result of `repr()`
should generally contain
more information than
the result of `str()`

`str()` is for *clients*



As a rule, you should
always write a `repr()`
for your classes

The default `repr()`
is not very helpful

The **built-in** function

str()

produces a **readable**, human-friendly
representation of an object

Remember - `str()`
is also the string
constructor

Not programmer
oriented



By default, `str()`
simply **calls** `repr()`



But `repr()` does **not** call `str()`



`repr()` is used when showing
elements of a collection

The **special** method

__format__()

invoked by `str.format()`



```
>>> obj = MyClass()
>>> "{:XYZ}".format(obj)
```



```
class MyClass:
```

```
    def __format__(self, f):
        # Return a string representation of self
        # formatted according to the formatting
        # options in format specifier f
        # ...
```



```
>>> obj = MyClass()
>>> "{:XYZ}".format(obj)
```

```
class MyClass:
```

```
    def __format__(self, f):
        # Return a string representation of self
        # formatted according to the formatting
        # options in format specifier f
        # ...
```



Replacement fields

{field_name:format_spec}

Optional **format specification**
after colon

The **standard library** module

reprlib

supports alternative
implementations of repr()

- limits otherwise excessive string length
- useful for large collections

The **standard library** function

`reprlib.repr()`

is a **drop-in replacement** for `repr()`

The **standard library** class

reprlib.Repr

- implements the main functionality of reprlib
- supports customisation through subclassing
- <http://docs.python.org/3/library/reprlib.html>

reprlib.aRepr

an **instance** used
by Python and
debuggers



The **built-in** functions

`ascii()`

`ord()` `chr()`

The **built-in** function

ascii()

replaces non-ASCII characters with escape sequences

The **built-in** function

chr()

Converts an **integer** Unicode
codepoint to a single character string

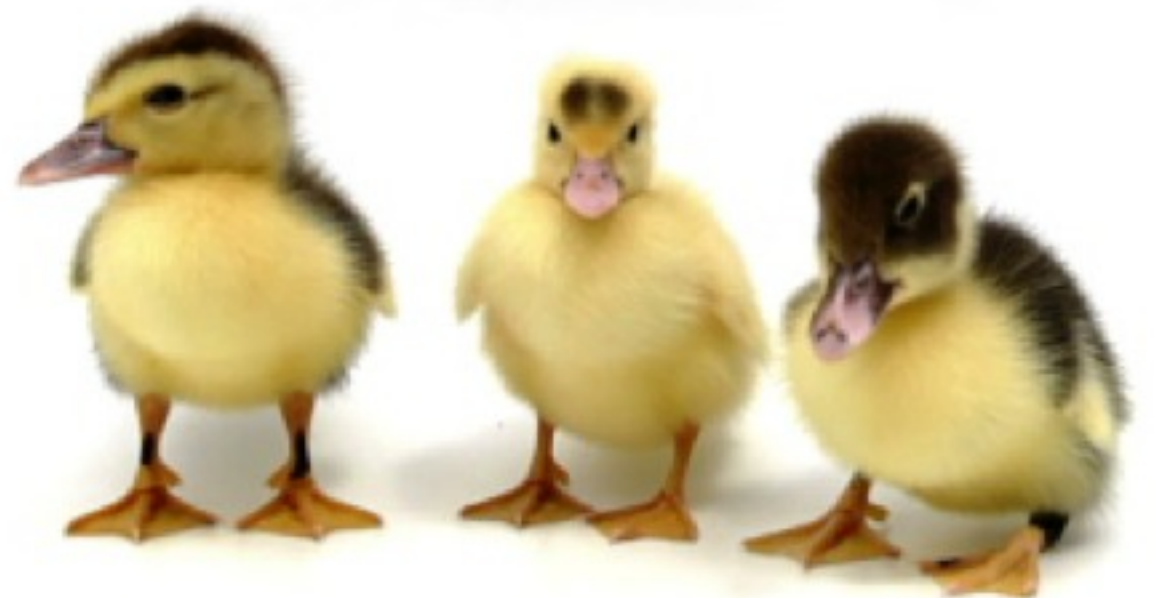
The **built-in** function

ord()

converts a single character to its
integer Unicode codepoint

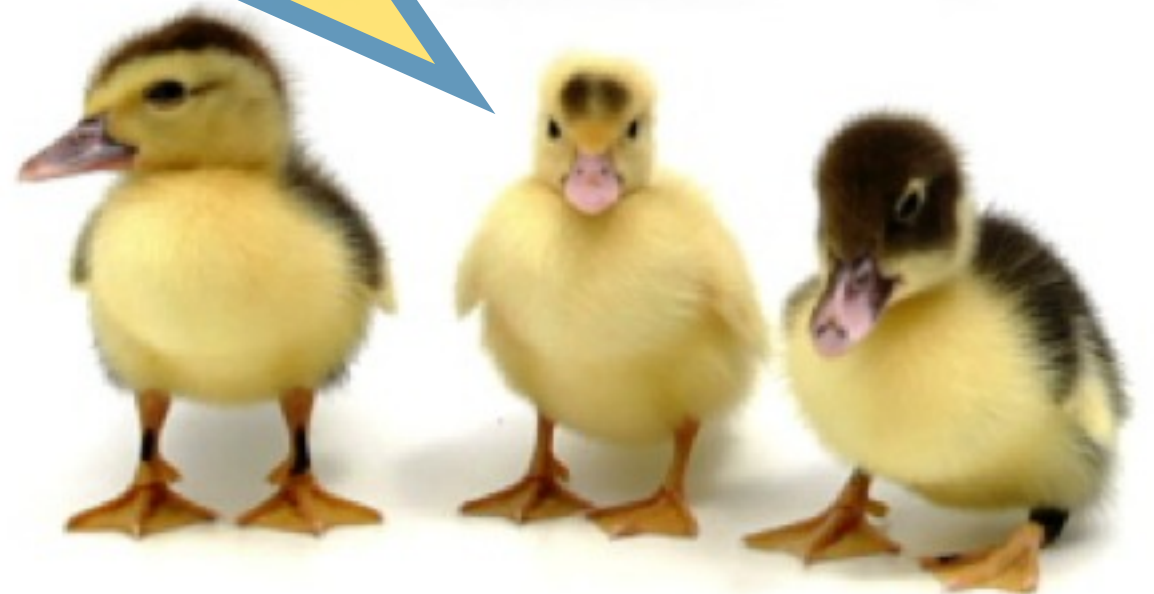
Duck Tails

Bigger Isn't Always Better



Duck Tails

SOMETIMES THE
`repr()` OF AN OBJECT WILL
BE SHORTER THAN
ITS `str()`



Strings and Representations

`print(obj)`



`str(obj)`



`repr(obj)`



`"{:f}".format(obj)`



`__str__(self)`

`__repr__(self)`

`__format__(self, f)`

an object

- for **humans!**
- fallback to `repr()`

- unambiguous
- precise
- include type
- for **developers**

- fallback to `str()`

```
import reprlib
```

```
reprlib.repr(obj)
```

```
from reprlib import Repr
```

```
class MyRepr(Repr):
```

```
    # ...
```

```
s = ascii(string)
```

```
i = ord(c)
```

```
c = chr(i)
```