Lambda functions

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Hugo Bowne-Anderson
Instructor

Lambda functions

```python
raise_to_power = lambda x, y: x ** y


raise_to_power(2, 3)
```

```
8
```

2. Lambda functions
There's a quicker way to write functions on the fly and these are called lambda functions because you use the keyword lambda. Here we re-write our function raise_to_power as a lambda function. To do so, after the keyword lambda, we specify the names of the arguments; then we use a colon followed by the expression that specifies what we wish the function to return. Lambda functions allow you to write functions in a quick and potentially dirty way so I wouldn't advise you to use them all the time but there are situations when they can come in very handy.

Anonymous functions

- Function map takes two arguments: `map(func, seq)`
- `map()` applies the function to ALL elements in the sequence

```
nums = [48, 6, 9, 21, 1]

square_all = map(lambda num: num ** 2, nums)

print(square_all)
```

```
<map object at 0x103e065c0>
```

```
print(list(square_all))
```

```
[2304, 36, 81, 441, 1]
```

3. Anonymous functions
For example, check out the map function, which takes two arguments, a function and a sequence such as a list and applies the function over all elements of the sequence. We can pass lambda functions to map without even naming them and in this case we refer to them as anonymous functions. In this example, we use map on a lambda function that squares all elements of a list and we'll store the result in square_all. Printing square_all reveals that it is actually a map object so to see what it contains we use the function list to turn it into a list and print the results to the shell. As expected, it's a list containing the squares of the elements in the original list!

Let's practice!

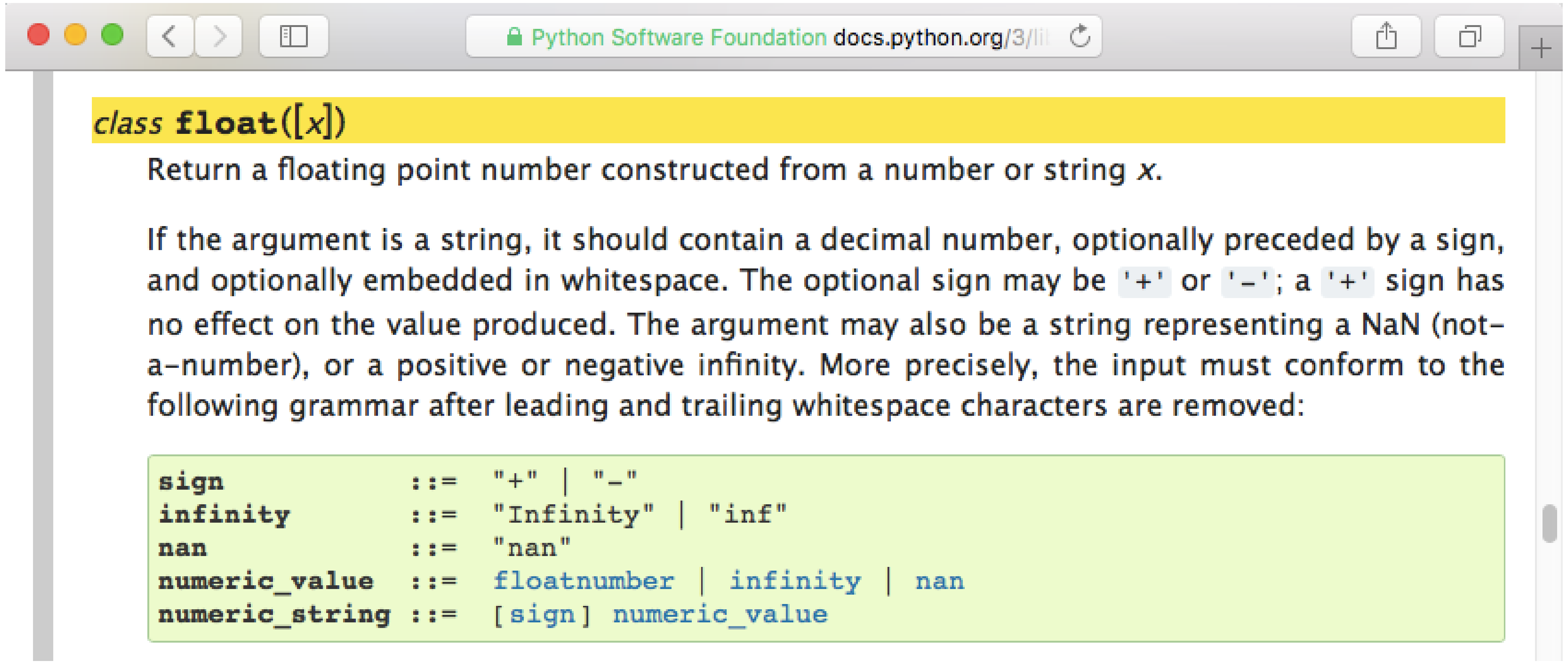
PYTHON DATA SCIENCE TOOLBOX (PART 1)

Introduction to error handling

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Hugo Bowne-Anderson
Instructor

The float() function



class **float**([*x*])

Return a floating point number constructed from a number or string *x*.

If the argument is a string, it should contain a decimal number, optionally preceded by a sign, and optionally embedded in whitespace. The optional sign may be `'+'` or `'-'`; a `'+'` sign has no effect on the value produced. The argument may also be a string representing a NaN (not-a-number), or a positive or negative infinity. More precisely, the input must conform to the following grammar after leading and trailing whitespace characters are removed:

```
sign           ::=  "+" | "-"
infinity       ::=  "Infinity" | "inf"
nan            ::=  "nan"
numeric_value  ::=  floatnumber | infinity | nan
numeric_string ::=  [sign] numeric_value
```

Passing an incorrect argument

```
float(2)
```

```
2.0
```

```
float('2.3')
```

```
2.3
```

```
float('hello')
```

```
<hr />-------------------------------------------------------------
ValueError                             Traceback (most recent call last)
<ipython-input-3-d0ce8bccc8b2> in <module>()
<hr />-> 1 float('hi')
ValueError: could not convert string to float: 'hello'
```

Passing valid arguments

```python
def sqrt(x):
    """Returns the square root of a number."""
    return x ** (0.5)
sqrt(4)
```

```
2.0
```

```python
sqrt(10)
```

```
3.1622776601683795
```

Passing invalid arguments

```
sqrt('hello')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-cfb99c64761f> in <module>()
----> 1 sqrt('hello')
<ipython-input-1-939b1a60b413> in sqrt(x)
      1 def sqrt(x):
----> 2     return x**(0.5)
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'float'
```

5. Passing invalid arguments
What happens if we pass it a string such as 'hello'? Then it throws me an error corresponding to a line of code within the function definition. This error says it was some sort of TypeError but the message may not be particularly useful to a user of our function, so we should endeavor to provide useful error messages for the functions we write.

Errors and exceptions

- Exceptions - caught during execution
- Catch exceptions with try-except clause
    - Runs the code following try
    - If there’s an exception, run the code following except

6. Errors and exceptions
This is an example of an error caught during execution, commonly called exceptions.
The main way to catch such exceptions is the try-except clause, in which Python tries to run the code following try and if it can, all is well. If it cannot due to an exception, it runs the code following except.

Errors and exceptions

```
def sqrt(x):
    """Returns the square root of a number."""
    try:
        return x ** 0.5
    except:
        print('x must be an int or float')

sqrt(4)
```

7. Errors and exceptions
Let's now rewrite our square root function but this time catch any exceptions raised. So here, we try to execute x to the power of zero point five; using except, in the case of an exception, we print 'x must be an int or float'. Now we see that the resulting function behaves well for ints and floats and also prints out what we wanted it to for a string.

```
2.0
```

```
sqrt(10.0)
```

```
3.1622776601683795
```

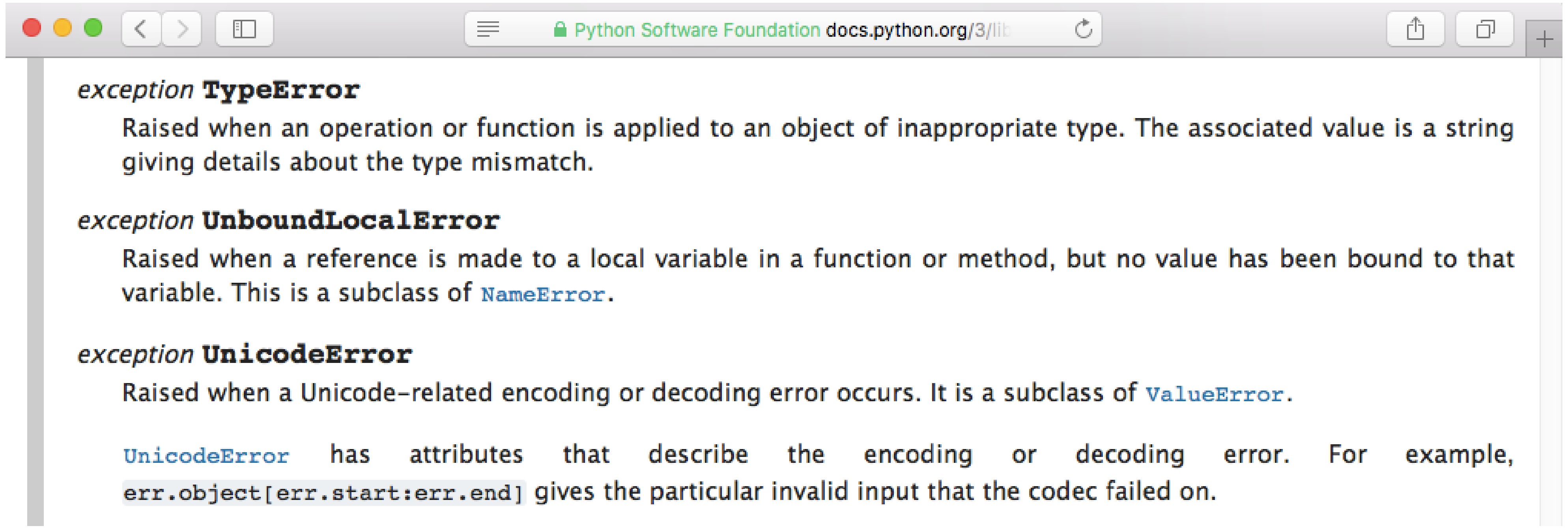
```
sqrt('hi')
```

```
x must be an int or float
```

Errors and exceptions

```python
def sqrt(x):
    """Returns the square root of a number."""
    try:
        return x ** 0.5
    except TypeError:
        print('x must be an int or float')
```

8. Errors and exceptions
We may also wish to only catch TypeErrors and let other errors pass through, in which case we would use except TypeError as you can see here. There are many other types of exceptions that can be caught and you can have a look at them in the Python documentation available online.



exception **TypeError**
Raised when an operation or function is applied to an object of inappropriate type. The associated value is a string giving details about the type mismatch.

exception **UnboundLocalError**
Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable. This is a subclass of `NameError`.

exception **UnicodeError**
Raised when a Unicode-related encoding or decoding error occurs. It is a subclass of `ValueError`.

`UnicodeError` has attributes that describe the encoding or decoding error. For example, `err.object[err.start:err.end]` gives the particular invalid input that the codec failed on.

Errors and exceptions

```
sqrt(-9)
```

```
(1.8369701987210297e-16+3j)
```

```
def sqrt(x):
    """Returns the square root of a number."""
    if x < 0:
        raise ValueError('x must be non-negative')
    try:
        return x ** 0.5
    except TypeError:
        print('x must be an int or float')
```

9. Errors and exceptions
More often than not, instead of merely printing an error message, we'll want to actually raise an error by using the keyword raise. For example, our square root function does something we may not desire when applied to negative numbers. It actually returns a complex number which we may not want. In fact, let's say that we don't wish our function to work for negative numbers. Then using an if clause, we can raise a ValueError for cases in which the user passes the function a negative number.

Errors and exceptions

```
sqrt(-2)
```

```
-----------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-2-4cf32322fa95> in <module>()
----> 1 sqrt(-2)
<ipython-input-1-a7b8126942e3> in sqrt(x)
      1 def sqrt(x):
      2     if x < 0:
----> 3         raise ValueError('x must be non-negative')
      4     try:
      5         return x**(0.5)
ValueError: x must be non-negative
```

10. Errors and exceptions
If we pass our new function a negative number, see it returns the prescribed ValueError!

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Bringing it all together

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Hugo Bowne-Anderson
Instructor

Errors and exceptions

```python
def sqrt(x):
    try:
        return x ** 0.5
    except:
        print('x must be an int or float')
```

```python
sqrt(4)
```

```
2.0
```

```python
sqrt('hi')
```

```
x must be an int or float
```

Errors and exceptions

```
def sqrt(x):
    if x < 0:
        raise ValueError('x must be non-negative')
    try:
        return x ** 0.5
    except TypeError:
        print('x must be an int or float')
```

In the following interactive exercises, you'll write error messages using two methods that you have learned: one, using the try-except syntax that you see here;

3. Errors and exceptions
two: explicitly raising errors using the keyword raise as in this example.

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Congratulations!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Hugo Bowne-Anderson
Instructor

What you've learned:

- Write functions that accept single and multiple arguments
- Write functions that return one or many values
- Use default, flexible, and keyword arguments
- Global and local scope in functions
- Write lambda functions
- Handle errors

There's more to learn!

- Create lists with list comprehensions
- Iterators - you've seen them before!
- Case studies to apply these techniques to Data Science

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)