

# Lambda functions

PYTHON DATA SCIENCE TOOLBOX (PART 1)



**Hugo Bowne-Anderson**  
Instructor

# Lambda functions

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

```
raise_to_power(2, 3)
```

```
8
```

# 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))
```

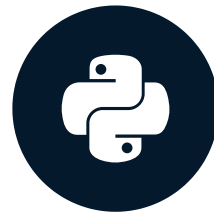
```
[2304, 36, 81, 441, 1]
```

# Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

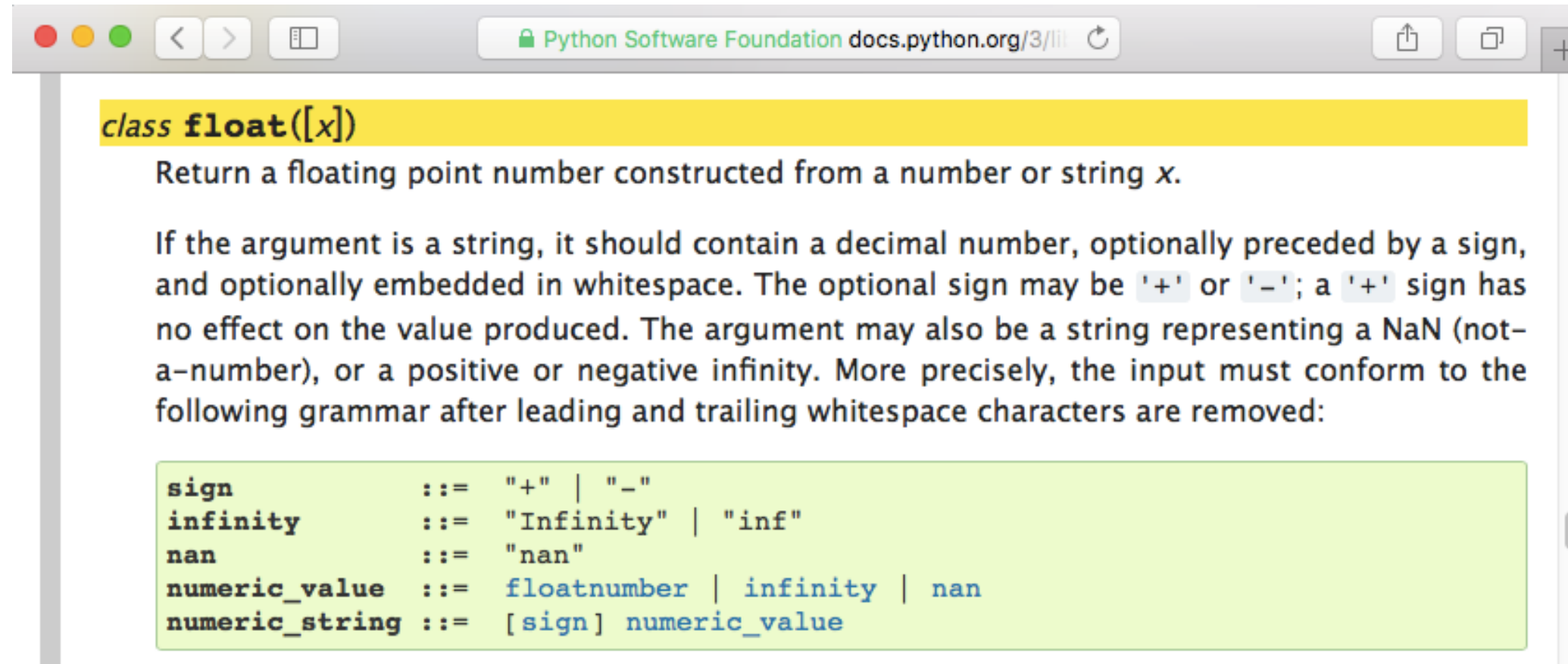
# Introduction to error handling

PYTHON DATA SCIENCE TOOLBOX (PART 1)



**Hugo Bowne-Anderson**  
Instructor

# The float() function



The screenshot shows a web browser window with the URL `docs.python.org/3/`. The page content is as follows:

**`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

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

```
2.0
```

```
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'
```

# 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

# 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)
```

```
2.0
```

```
sqrt(10.0)
```

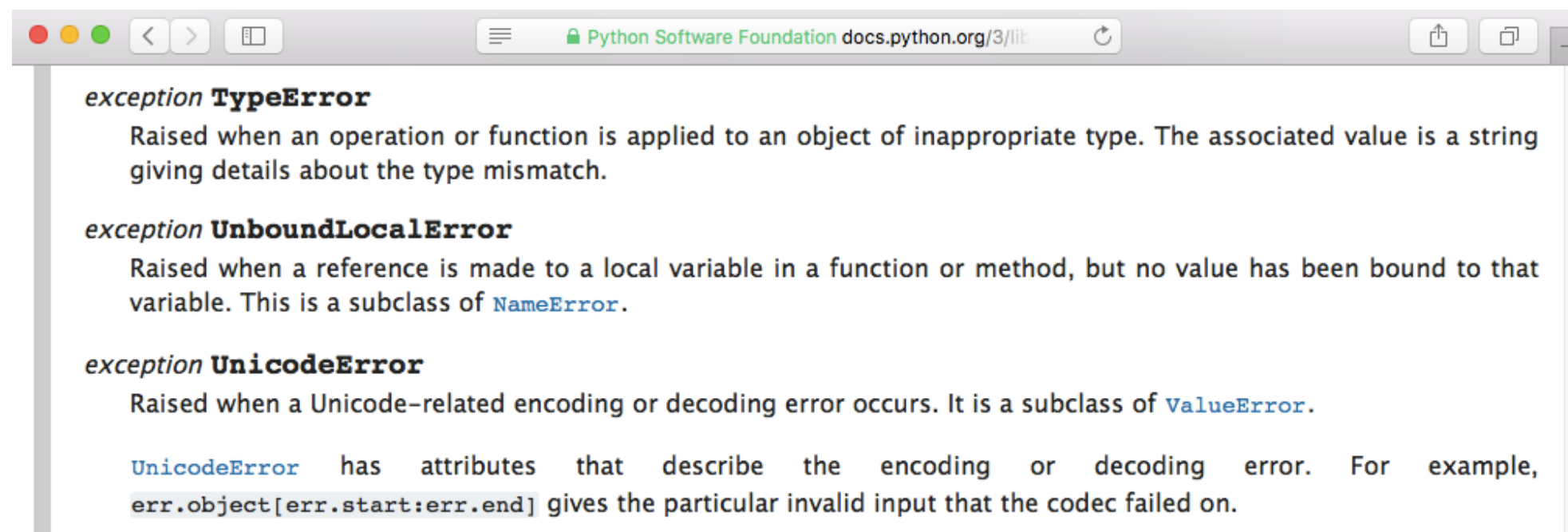
```
3.1622776601683795
```

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

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

# Errors and exceptions

```
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')
```



# 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')
```

# 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
```

# 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

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

```
sqrt(4)
```

```
2.0
```

```
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')
```

# 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)