Error Handling: Takeaways

Syntax

SETS

- Transforming a list into a set:

```
animals = ["Dog", "Cat", "Hippo", "Dog", "Cat", "Dog", "Dog", "Cat"]
unique_animals = set(animals)
```

- Displaying the set:

```
`{'Hippo', 'Cat', 'Dog'}`
```

- Adding items to a set:

```
unique_animals.add("Tiger")
```

- Removing items from a set:

```
unique_animals.remove("Dog")
```

- Converting a set into a list:

```
list(unique_animals)
```

DEALING WITH MISSING VALUES AND PARSING DATA

- To replace missing values in the party column:

```
rows = [
    ["Bassett", "Richard", "1745-04-02", "M", "sen", "DE", "Anti-Administration"],
    ["Bland", "Theodorick", "1742-03-21", "", "rep", "VA", ""]
]
for row in rows:
    if row[6] == "":
        row[6] = "No Party"
```

- To split a date into its component parts:

```
date = "1820-01-02"

parts = date.split("-")

print(parts)
```

TRY/EXCEPT BLOCKS

- To handle our error and allow our code to run, use a try/except block:

```
try:
    int('')
except Exception:
    print("There was an error")
```

- To add an exception instance:

```
try:
    int('')
except Exception as exc:
    print(type(exc))
```

- To convert the Exception class to a *string* and print out the error message:

```
try:
    int('')
except Exception as exc:
    print(str(exc))
```

USING THE PASS KEYWORD

- When we don't want anything in the except statement body, we can use pass:

```
try:
    int('')
except Exception:
    pass
```

CONVERT BIRTH YEARS TO INTEGERS

- To convert a column within a list of lists from a string to integer:

```
items = [
    [1, "1", 2],
    [2, "", 3],
    [5, "5", 3]
]
for item in items:
    item[1] = int(item[1])
```

FILL IN YEARS WITHOUT A VALUE

- To replace invalid values in our list of lists:

```
last_value = 1
for row in legislators:
    if row[7] == 0:
        row[7] = last_value
    last_value = row[7]
```

Concepts

- A **set** is a data type, similar to a list, except each element is unique. If you add an element to a set that it already contains, the set will ignore it.
- When exploring a dataset, it's always a good idea to look for:
    - Missing data
    - Values that don't make intuitive sense
    - Recurring themes
- To address missing data:
    - Remove any rows with missing data
    - Populate empty field with specified values

- Populate empty field with specified values
- Populate empty fields with calculated value
- Use analysis techniques to work with missing data

Resources

- Python Documentation on Error Handling