MODULE 2

Data types

DATA TYPES

- Every value has a type, and the built-in ***type*** function returns the type of the result of any expression.

- Examples:
  - type(abs) – inbuilt method or function
  - type(1) - int
  - a = 2 : type(a) – int
  - Type(3.245) -- float

BUILT IN FUNCTION OR METHOD

- Example of inbuilt function is abs.

- We can check its type using:

```
type(abs)
```

```
builtin_function_or_method
```

(Demo) notebook 2.1

NUMBERS

INTS AND FLOATS

- Python has two real number types
  - **int**: an integer of any size
  - **float:** a number with an optional fractional part

- An **int** never has a decimal point; a **float** always does
- A **float** might be printed using scientific notation
- Three limitations of float values:
  - They have limited size (but the limit is huge)
  - They have limited precision of 15-16 decimal places
  - After arithmetic, the final few decimal places can be wrong

(Demo) notebook 2.1

STRINGS

TEXT AND STRINGS

- A string value is a snippet of text of any length
  - 'a'
  - 'word'
  - "there can be 2 sentences. Here's the second!"
- Strings that contain numbers can be converted to numbers
  - int('12')
  - float('1.2')
- Any value can be converted to a string
  - str(5)

(Demo) notebook 2.1

EXAMPLES OF STRING METHODS

- upper() – turns string into upper case
  - E.g; “loud”.upper() -- LOUD
- lower() – turns string into lower case
  - E.g; “loud”.lower() -- loud
- capitalize() – capitalizes the first letter of the string
  - E.g; “loud”.capitalize() – Loud
- replace() – replaces a substring of the string with another string
  - E.g; “loud”.replace(”lo”, “clo”) -- cloud

(Demo)

DISCUSSION QUESTION

Assume you have run the following statements

```
x = 3
y = '4'
z = '5.6'
```

What's the source of the error in each example?

A. `x + y`
B. `x + int(y + z)`
C. `str(x) + int(y)`
D. `str(x, y) + z`

COMPARISONS

BOOLEANS

- Boolean values most often arise from comparison operators.
  - Ex. 1 – comparing numbers

```
3 > 1 + 1
```

```
True
```

COMPARING STRINGS

- When comparing strings, we consider their order alphabetically.
- A shorter string is less than a longer string that begins with the shorter string.
  - Example:

```
"Dog" > "Catastrophe" > "Cat"
```

```
True
```

MOST COMMON PYTHON COMPARISON OPERATORS

- Python includes a variety of operators that compare values.

Comparison	Operator	True example	False Example
Less than	<	2 < 3	2 < 2
Greater than	>	3>2	3>3
Less than or equal	<=	2 <= 2	3 <= 2
Greater or equal	>=	3 >= 3	2 >= 3
Equal	==	3 == 3	3 == 2
Not equal	!=	3 != 2	2 != 2

SEQUENCES

1. ARRAYS

An array contains a sequence of values

- All elements of an array should have the same type
- Arithmetic is applied to each element individually
- When two arrays are added, they must have the same size; corresponding elements are added in the result
- A column of a table is an array

(Demo) – notebook 2.1

2. RANGES

A range is an array of consecutive numbers

- **np.arange(end)**:
  An array of increasing integers from 0 up to **end**
- **np.arange(start, end)**:
  An array of increasing integers from **start** up to **end**
- **np.arange(start, end, step)**:
  A range with **step** between consecutive values

The range always includes **start** but excludes **end**

(Demo) – notebook 2.1

LISTS

LISTS ARE GENERIC SEQUENCES

A list is a sequence of values (just like an array), but the values can all have different types

```
[2+3, 'four', Table().with_column('K', [3, 4])]
```

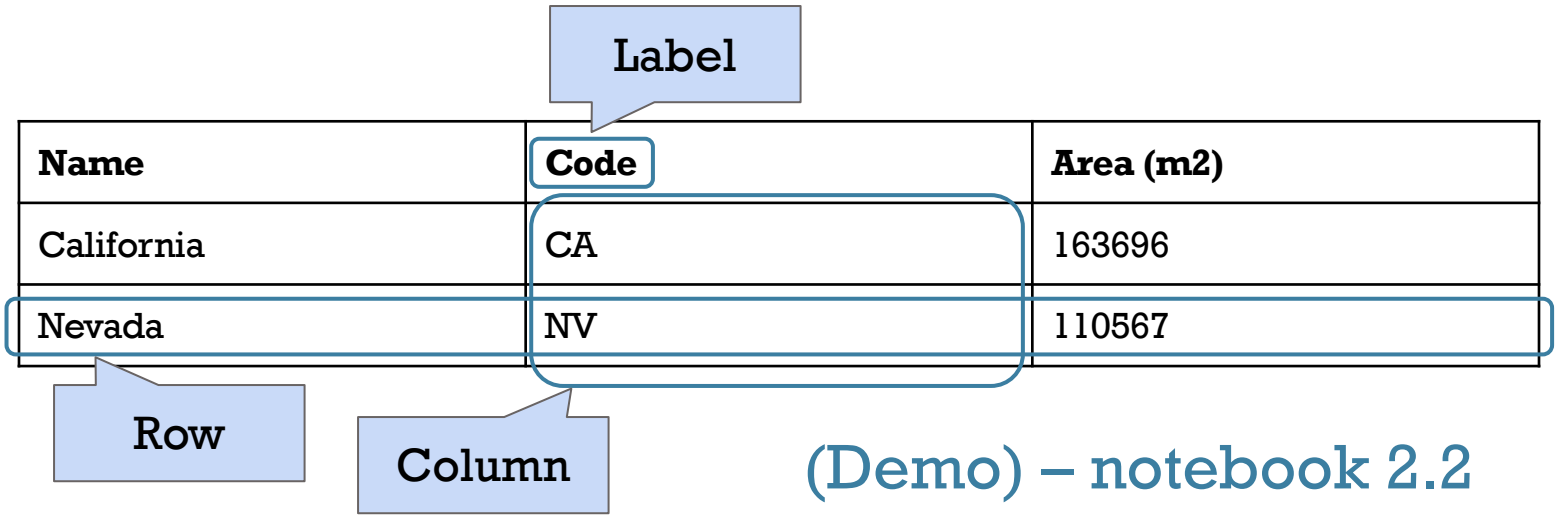
If you create a table column from a list, it will be converted to an array automatically

(Demo) – notebook 2.1

TABLES

TABLE STRUCTURE

- We organize our data in tables
- A Table is a sequence of labeled columns
- Data within a column should be of the same "type"



(Demo) – notebook 2.2

WAYS TO CREATE A TABLE

- `Table()` - an empty table
- `Table.read_table(filename)` - reads a table from a spreadsheet
- and…

(Demo) – notebook 2.2

ARRAYS → TABLES

- `Table().with_column(label, data)` - creates a table with a single column; `data` is an array
- `Table().with_columns(label1, data1, ...)` - creates a table, with an array of data for each column

(Demo) – notebook 2.2

THE WHERE METHOD

- `t.where(label, condition)` - constructs a new table with just **the rows that match** the condition

(Demo) – notebook 2.2

TAKE ROWS, SELECT COLUMNS

The `take` method returns a table with only some rows

- Rows are numbered, starting at 0
- Taking a single number returns a one-row table
- Taking a list of numbers returns a table as well

The `select` method returns a table with only some columns

(Demo) – notebook 2.2

TABLE OPERATIONS

- `t.select(label)` - constructs a new table with just the specified columns
- `t.sort(label)` - constructs a new table, with rows sorted by the specified column

(Demo) – notebook 2.2

TABLE METHODS

- Creating and extending tables:
  - `Table().with_columns` and `Table.read_table`
- Finding the size: `num_rows` and `num_columns`
- Referring to columns: labels, relabeling, and indices
  - `labels` and `relabeled`; column indices start at 0
- Accessing data in a column
  - `column` takes a label or index and returns an array
- Using array methods to work with data in columns
  - `item`, `sum`, `min`, `max`, and so on
- Creating new tables containing some of the original columns:
  - `select`, `drop`

EXAMPLES

The table `students` has columns `Name`, `ID`, and `Score`.
Write one line of code that evaluates to:

a) A table consisting of only the column labeled `Name`

```
students.select('Name')
```

b) The largest score

```
students.column('Score').max()
max(students.column('Score'))
```

MANIPULATING ROWS

- `t.sort(column)` sorts the rows in increasing order
- `t.take(row_numbers)` keeps the numbered rows
  - Each row has an index, starting at 0
- `t.where(column, are.condition)` keeps all rows for which a column's value satisfies a condition
- `t.where(column, value)` keeps all rows for which a column's value equals some particular value
- `t.with_row` makes a new table that has another row

CENSUS DATA

THE DECENNIAL CENSUS

- Every ten years, the Census Bureau counts how many people there are in the U.S.
- In between censuses, the Bureau estimates how many people there are each year.
- Article 1, Section 2 of the Constitution:
  - "Representatives and direct Taxes shall be apportioned among the several States … according to their respective Numbers …"

ANALYZING CENSUS DATA

Leads to the discovery of interesting features and trends in the population

(Demo)

CENSUS TABLE DESCRIPTION

- Values have column-dependent interpretations
  - The SEX column: 1 is *Male*, 2 is *Female*
  - The POPESTIMATE2010 column: *7/1/2010 estimate*
- In this table, some rows are sums of other rows
  - The SEX column: 0 is *Total* (of *Male* + *Female*)
  - The AGE column: 999 is *Total* of all ages
- Numeric codes are often used for storage efficiency
- Values in a column have the same type, but are not necessarily comparable (AGE 12 vs AGE 999)

http://www2.census.gov/programs-surveys/popest/datasets/2010-2015/national/asrh/nc-est2015-agesex-res.pdf

DISCUSSION QUESTIONS

The table **nba** has columns **NAME**, **POSITION**, and **SALARY**.

a) Create an array containing the names of all point guards (**PG**) who make more than $15M/year

```
nba.where(1, 'PG').where(2, are.above(15)).column(0)
```

b) After evaluating these two expressions in order, what's the result of the second one?

```
nba.with_row(['Samosa', 'Mascot', 100])
nba.where('NAME', are.containing('Samo'))
```

QUESTIONS