Bertelsmann Tech Scholarship - **Data Track**

LESSON 21 Introduction to Data Analysis & Programming

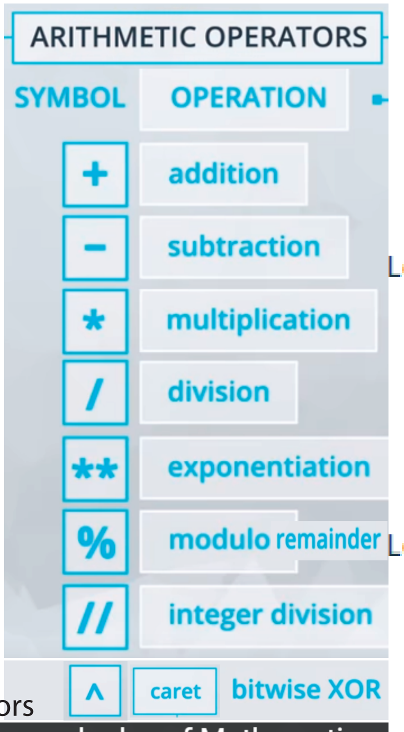
Why Python Programming

LESSON 22

Data Types and Operators

Familiarize yourself with the building blocks of Python! Learn about data types and operators, compound data structures, type conversion, built-in functions, and style guidelines.

22.2 Arithmetic Operators
- Order of Operations = BEDMAS



ARITHMETIC OPERATORS

SYMBOL	OPERATION
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation
%	modulo remainder
//	integer division
^ caret	bitwise XOR

Bitwise operators are special operators in Py
https://wiki.python.org/moin/BitwiseOperators

an arithmetic operator that doesn't follow the usual rules of Mathematics.

22.5 Variables and Assignment Operators
- Pythonic way to name variables is to use all lowercase letters and under-scores to separate words, called snake case --> my_height = 58

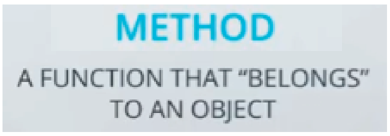
22.8 Integers and Floats
- type() --> built in fn, return type of obj >>> print(type(x))
- float can say a range of number = 0.3333

22.10 Booleans, Comparison Operators, Logical Operators
- 6 comparison operators:
  < Less Than; > Greater Than;
  <= Less Than or Equal To;
  >= 5 Greater Than or Equal To;
  == Equal To; != 5 Not Equal To
- 3 logical operators: and - or - not

22.13 Strings

22.16 Type and Type Conversion
- Diff types have diff behaviors, properties.

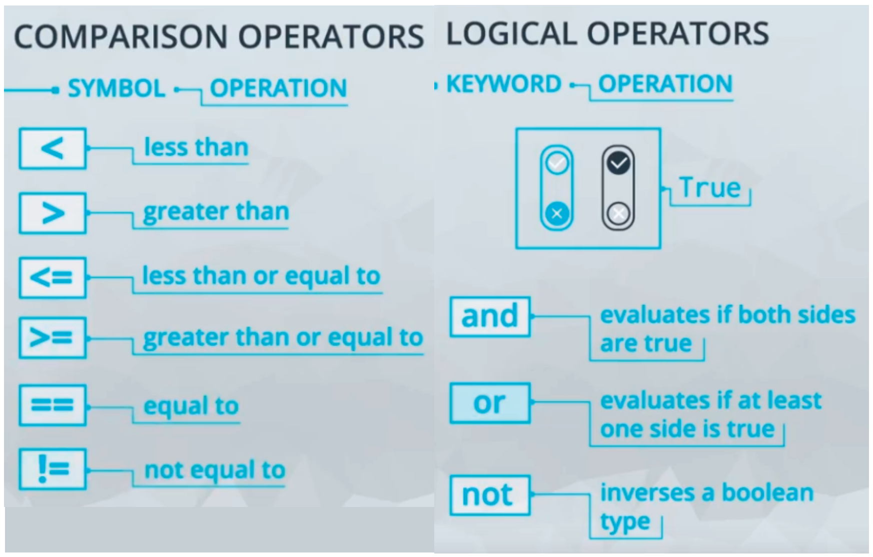
22.19
- 2 ways to process data: w/ operator & fn.
- operator = symbol
- fn = name, put data in ()
- method relate to fn, associated w/ special obj
- methods are fns that belongs to an obj (str)



METHOD

A FUNCTION THAT "BELONGS" TO AN OBJECT

obj
my_string = "today friday"
>>> my_string.count('a')
2
function method



COMPARISON OPERATORS

SYMBOL	OPERATION
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	not equal to

LOGICAL OPERATORS

KEYWORD	OPERATION
	True
and	evaluates if both sides are true
or	evaluates if at least one side is true
not	inverses a boolean type

Lesson 2: Data Types and Operators
- Data Types: Integers, Floats, Booleans, Strings, Lists, Tuples, Sets, Dictionaries
- Operators: Arithmetic, Assignment, Comparisn, Logicl, Membershp, Identty
- Built-In Functions, Compound Data Structures, Type Conversion
- Whitespace and Style Guidelines

Lesson 3: Control Flow
- Conditional Statements
- Boolean Expressions
- For and While Loops
- Break and Continue
- Zip and Enumerate
- List Comprehensions

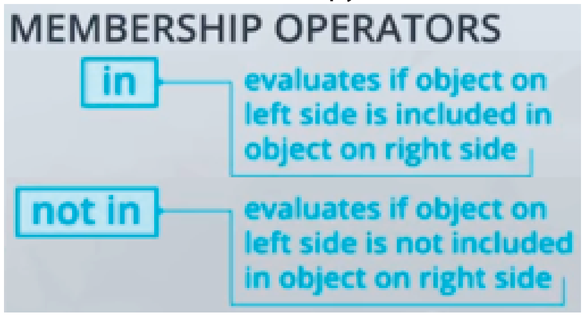
Lesson 4: Functions
- Defining Functions
- Variable Scope
- Documentation
- Lambda Expressions
- Iterators and Generators

Lesson 5: Scripting
- Python Installation and Environment Setup
- Running and Editing Python Scripts
- Interacting with User Input
- Handling Exceptions
- Reading and Writing Files
- Importing Local, Standard, and Third-Party Modules
- Experimenting with an Interpreter

22.21 List and Membership Operators

--> **Lists! = Data structures** = containers that organize and group data types together in different ways.

list_of_random_things = [1, 3.4, 'a string', True] // this is a list of 4 elements. lists are indexed in py



MEMBERSHIP OPERATORS

in — evaluates if object on left side is included in object on right side

not in — evaluates if object on left side is not included in object on right side

>>> list_of_random_things[0]
1
>>> list_of_random_things[len(list_of_random_things) - 1]
True // can index from the end of a list by using negative values, -1
// > last element, -2 > second to last ele.. so on.
>>> list_of_random_things[-1]
True

--> **Slice and Dice with Lists**

- Pull more than 1 value from a list at a time by using slicing, lower index is inclusive & upper index is exclusive.

>>> list_of_random_things = [1, 3.4, 'a string', True]
>>> list_of_random_things[1:2]
[3.4]

--> **Are you in OR not in?**

- use in and not in to return a bool of whether an element exists within our list.

>>> 'this' in 'this is a string'
True

--> **Mutability and Order**

Mutability is about whether or not we can change an object once it has been created.

- List can be changed = mutable. Strings is considered immutable, we need to create a completely new string.
- Order is about whether the position of an element in the object can be used to access the element. Strings and lists are ordered. We can use the order to access parts of a list and string.

22.23 List Methods

Useful Functions for Lists 1:

--> len() returns how many elements are in a list.
--> max() returns the greatest element of the list > depends on what type objects are in the list. (numbers is the largest number).(list of strings is element were sorted alphabetically) max function is defined by greater than comparison operator (>). max fn is undefined for lists that contain eles from different, incomparable types.
--> min() returns smallest ele in a list. min = opposite of max.
--> sorted() returns a copy of a list in order from smallest to largest, leaving the list unchanged.

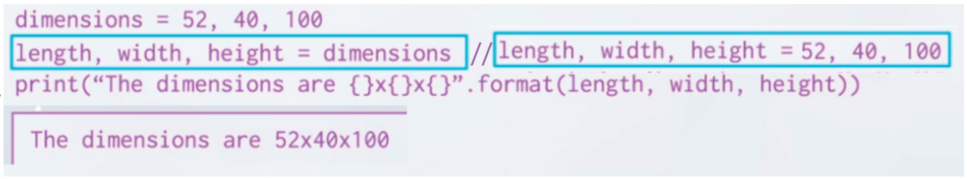
Useful Functions for Lists 2:

--> join() = string method that takes a list of strings as an argument, returns a string consisting of the list elements joined by a separator string // new_str = "\n".join(["fore", "aft", "starboard", "port"])
// "\n" = separator = a newline between each element.
--> to separate each items in the list you are joining with a comma (,)

22.25 Tuples

- Tuple = container = data type for immutable ordered sequences of elements, used to store related pieces of information.

location = (13.4125, 103.866667)
print("Latitude:", location[0])
print("Longitude:", location[1])



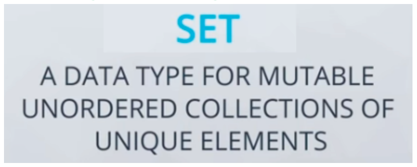
```
dimensions = 52, 40, 100
length, width, height = dimensions //length, width, height = 52, 40, 100
print("The dimensions are {}x{}x{}".format(length, width, height))
```

```
The dimensions are 52x40x100
```

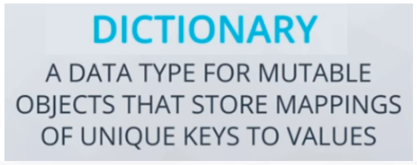
Tuples can also be used to assign multiple variables in a compact way.

- Tuples = lists = store an ordered collection of objs which can be accessed by their indices. However, tuples are immutable - you can't add and remove items from tuples, or sort them in place.
- Parentheses are optional when defining tuples, and frequently omit them if parentheses don't clarify the code.
- 3 vars are assigned from the content of the tuple dimensions, called tuple unpacking. You can use tuple unpacking to assign the information from a tuple into multiple variables without having to access them one by one and make multiple assignment statements. If we won't need to use dimensions directly, we could shorten those two lines of code into a single line that assigns three variables in one go!

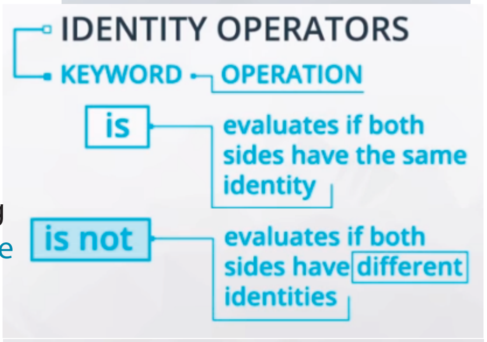
22.27 SETS = data type for mutable unordered collections of unique elements.
set remove duplicates from a list.
>>> numbers = [1, 2, 6, 3, 1, 1, 6]
>>> unique_nums = set(numbers) // {1, 2, 3, 6}


SET
A DATA TYPE FOR MUTABLE UNORDERED COLLECTIONS OF UNIQUE ELEMENTS

- Sets support in operator same as lists do. use add method, and remove elements use pop method, similar to lists. However, pop an ele from a set, a random ele is removed. sets, unlike lists = unordered, so no "last element".
- Other operations can perform w/ sets include mathematical sets. Methods like union, intersection, and difference are perform w/ sets faster w/ other containers.


DICTIONARY
A DATA TYPE FOR MUTABLE OBJECTS THAT STORE MAPPINGS OF UNIQUE KEYS TO VALUES

22.29 Dictionaries and Identity Operators
- set = simple data structure, collect unique eles.
- Dictionary = flexible, store pair of ele > key & value
- dict = mutable data type that stores mappings of unique keys to values.
>>> elements = {"hydrogen": 1, "helium": 2, "carbon": 6}   # dictionary
- Dicts' method, . get look up value in a dict, return None if the key isn't found.
>>> print(elements.get("dilithium")) # dilithium n't in dict, None return by get
- get = a better tool > normal square bracket lookups bec' errors can crash prog
>>> elements.get('kryptonite', 'There\'s no such element!') # return by yr choice


IDENTITY OPERATORS
KEYWORD
OPERATION
is
evaluates if both sides have the same identity
is not
evaluates if both sides have different identities

Identity Operators:
- check if a key returned None w/ is operator.
- check for the opposite using is not.

```
n = elements.get("dilithium")
print(n is None)        # True
print(n is not None)  # False
```

22.30 Quiz: Dictionaries and Identity Operators
- mutable   = list - immutable = str, int, float

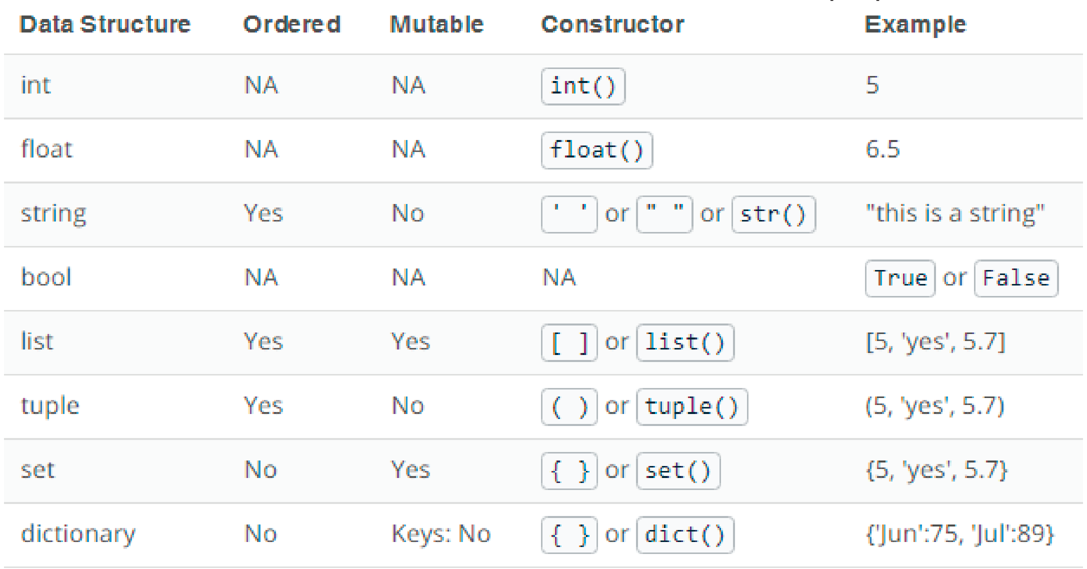
COMPARISON OPERATORS
!=
==

Checking for Equality vs. Identity: == vs. is
- List a, list b are equal & identical. List c equal to a (& b for that matter) since they have same contents. But a & c (& b for that matter, again) point to 2 different obj, i.e., they aren't identical obj. That is difference betw checking for equality vs. identity.

```
a = [1, 2, 3]
b = a
c = [1, 2, 3]
print(a == b)
print(a is b)
print(a == c)
print(a is c)
```

element =
```
{"hydrogen": {"number": 1,
              "weight": 1.00794,
              "symbol": 'H'},
   "helium": {"number": 2,
              "weight": 4.002602,
              "symbol": "He"},
   "oxygen": {"number": 8,
              "weight": 15.999,
              "symbol": "O"}}
```

22.33 Compound Data Structures
- Include containers in other containers to create compound data structures.
>> helium = elements["helium"]   # get the helium dict
>> hydrogen_weight = elements["hydrogen"]["weight"] # get hydrogen's wght
22.34 Quiz: Compound Data Structures
- Collections = group of data (elements). Data structures collections for storing, accessing, manipulating (lists, sets, dictionaries)- List sortable, add item to a list w/ .append, list item are always index w/ numbers starting at 0- Sets not ordered, inconsistent, add items to sets w/ .add. Like dict & lists, sets are mutable- Cann't have same item twice, cann't sort sets. For these 2 properties a list will be more appropriate.
- Each item in dict has 2 parts (key:value), items not ordered so not sortable. And not add items to dict w/ .append., & nested dict.
22.37 Summary
- Data Structures
- Mathematical, Comparison,
  and Logical Operators
- How George Boole's zeroes and ones changed the world =



Data Structure	Ordered	Mutable	Constructor	Example
int	NA	NA	`int()`	5
float	NA	NA	`float()`	6.5
string	Yes	No	`' '` or `" "` or `str()`	"this is a string"
bool	NA	NA	NA	`True` or `False`
list	Yes	Yes	`[ ]` or `list()`	[5, 'yes', 5.7]
tuple	Yes	No	`( )` or `tuple()`	(5, 'yes', 5.7)
set	No	Yes	`{ }` or `set()`	{5, 'yes', 5.7}
dictionary	No	Keys: No	`{ }` or `dict()`	{'Jun':75, 'Jul':89}

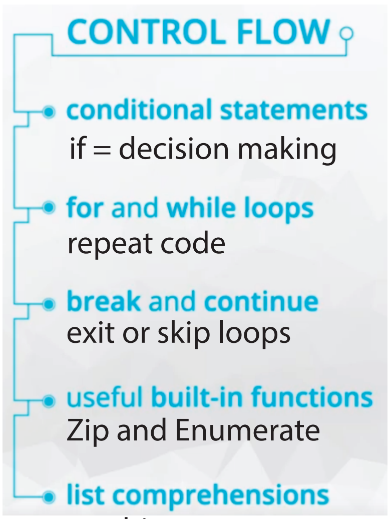
https://www.irishtimes.com/news/science/how-george-boole-s-zeroes-and-ones-changed-the-world-1.2014673

Bertelsmann Tech Scholarship - Data Track

LESSON 23 Introduction to Data Analysis & Programming

Control Flow

Build logic into your code with control flow tools! Learn about conditional statements, repeating code with loops and useful built-in functions, and list comprehensions.



CONTROL FLOW

conditional statements
if = decision making

for and while loops
repeat code

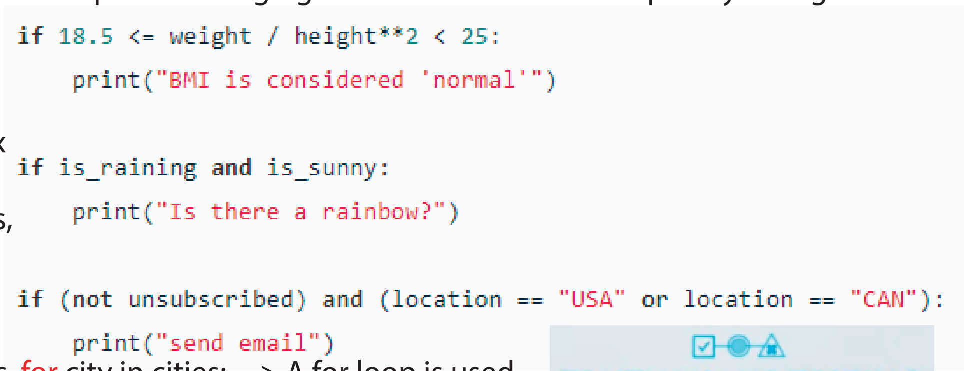
break and continue
exit or skip loops

useful built-in functions
Zip and Enumerate

list comprehensions
combine concepts

23.1 Intro Control Flow
Control flow is the sequence in which your code is run. Several tools in Python use to affect our code: bring data types & operators together.
- Conditional Statements > if = decision making
- Boolean Expressions
- For and While Loops > repeat code
- Break and Continue > exit or skip loops
- Zip and Enumerate > built in functions
- List Comprehensions > combine concepts

23.2 Conditional Statements
An if statement is a conditional statement that runs or skips code based on whether a condition is true or false. If, Elif, Else. Indentation > other languages use braces to show where blocks of code begin and end, Python use indentation to enclose blocks of code, if use indentation to tell Python what code is inside and outside of different clauses. Conventionally come in multiples of four spaces. Changing the indentation can completely change the meaning of the code.

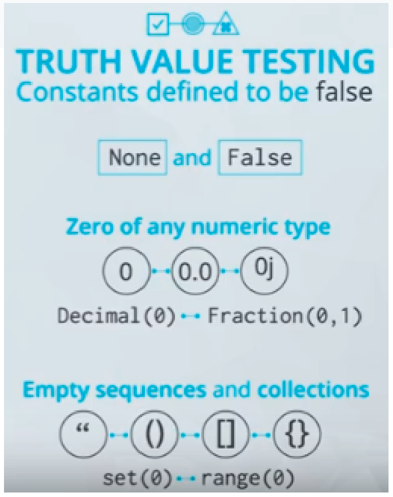


```
if 18.5 <= weight / height**2 < 25:
    print("BMI is considered 'normal'")

if is_raining and is_sunny:
    print("Is there a rainbow?")

if (not unsubscribed) and (location == "USA" or location == "CAN"):
    print("send email")
```

23.7 Boolean Expressions for Conditions
Complex Boolean Expressions
- If statements sometime use more complex boolean expressions for their conditions. May contain multiple comparison operators, logical operators, and even calculations.
- Truth Value Testing



TRUTH VALUE TESTING
Constants defined to be false

None and False

Zero of any numeric type
0 · 0.0 · 0j
Decimal(0) · Fraction(0,1)

Empty sequences and collections
'' · () · [] · {}
set(0) · range(0)

23.10 For Loops
- 2 kinds of loops - for loops and while loops. for city in cities: --> A for loop is used to "iterate" (city), or do some-thing repeatedly, over an iterable (cities). An iterable is an object that can return one of its elements at a time.
- Using the range() a built-in Function with for Loops -> used to create an iterable sequence of numbers, to repeat an action a certain number of times.
- range() function takes 3 integer arguments, range(start=0, stop, step=1), 1st and 3rd of which are optional.
- Creating and Modifying Lists w/ for loops, by appending to a new list at each iteration of for loop.

```
cities = ['new york city', 'mountain view', 'chicago', 'los angeles']
capitalized_cities = []
for city in cities:
    capitalized_cities.append(city.title())
```

Modifying a list requires range() function to generate indices for each value in cities list. This lets us access the elements of the list w/ cities[index] so that we can modify values in cities list in place.

```
cities = ['new york city', 'mountain view', 'chicago', 'los angeles']
for index in range(len(cities)):
    cities[index] = cities[index].title()
```

23.14 Iterating Through Dictionaries(key:value) with For Loops
>> for n in some_dict // access to the keys in the dictionary
>> to iterate through both keys, values in dict. > use built-in method items (returns tuples of key, value pairs,)

```
for key, value in cast.items():
    print("Actor: {}    Role: {}".format(key, value))
```