Introduction to Python (Part 2)

COMP 8347

Slides prepared by Dr. Arunita Jaekel

arunita@uwindsor.ca

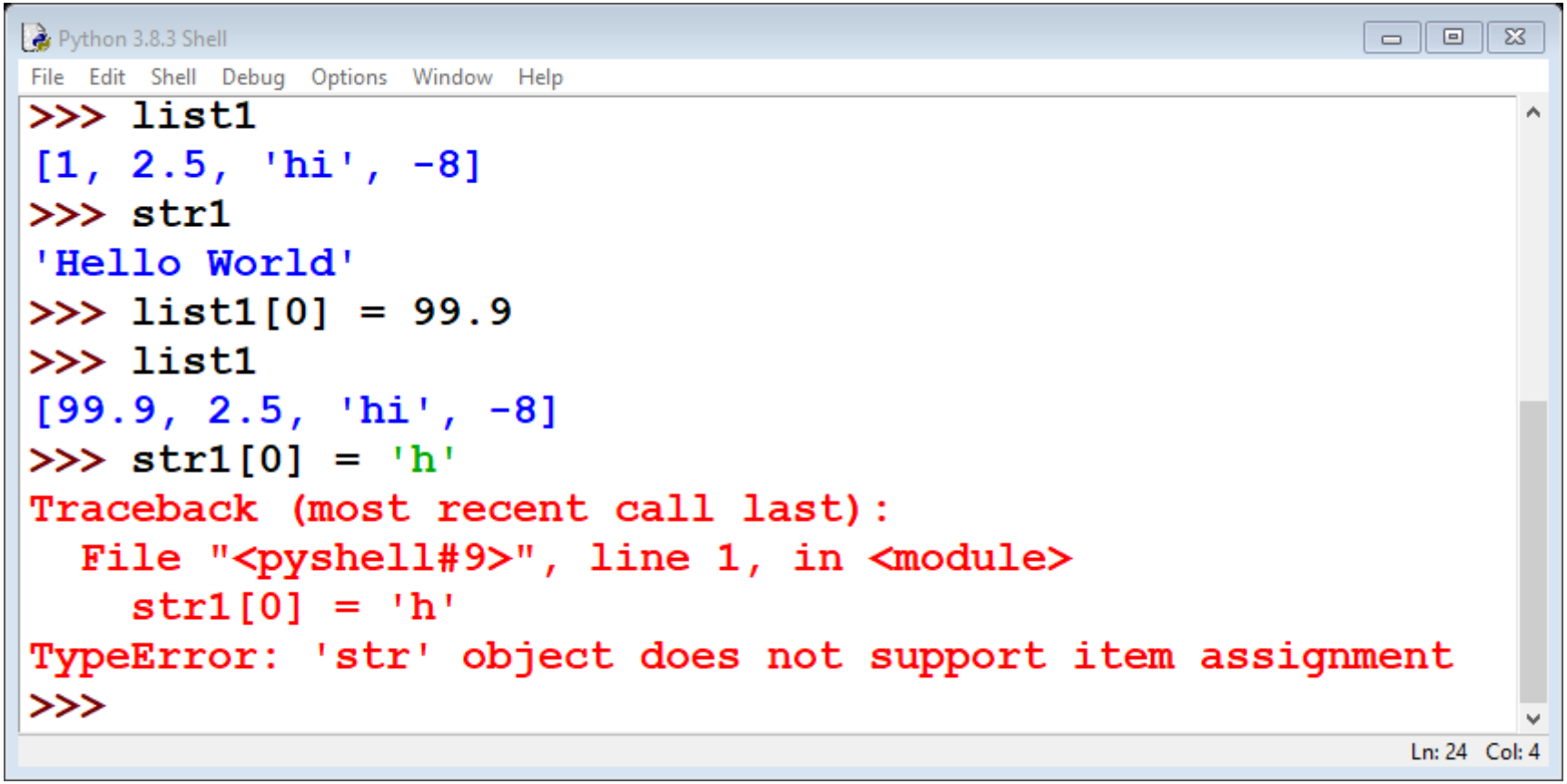
Python Basics

- Topics
  - Introduction
  - Collection Data Types
    - Strings
    - Lists
    - Dicts
  - Comparison/Logic Operations

Collection Data Types

- Holds a collection of items, which may or may not be of the same type.
- May be *mutable* (e.g. list, set, dict) or *immutable* (e.g. tuple, str)
- *sequence*: a type that supports membership(in), size function (len()), slices ([]) and is iterable. e.g. list, str, tuple
  - https://docs.python.org/3/tutorial/datastructures.html

Mutable vs Immutable

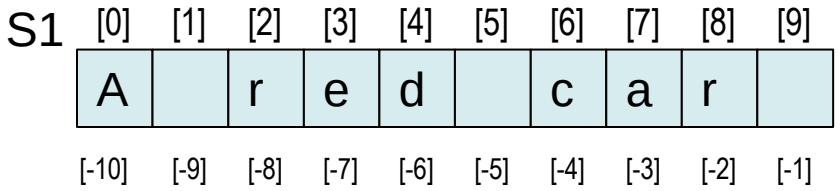


Strings

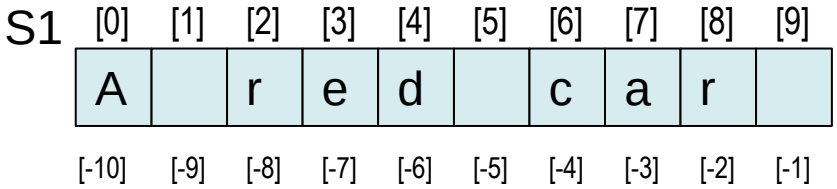
- A collection data type that is ***ordered*** and ***unchangeable*** (***immutable***).
  - e.g. [1], [3, ‘hi’, 4.5, [1,2,3]], []
- The string type in Python is called **str**
- String literals delimited by single or double quotes; e.g. ‘hello’ or “hello”
- Access individual items using the index number and enclosing in square brackets, e.g. str1[0]
- The first element always has index 0

Strings

- Use index([]) and ‘slice’ similar to lists
  - S1 = “A red car ”
  - >>> S1[4]
    - **‘d’**
  - >>> S1[:7]
    - **‘A red c’**
  - >>> S1[2:-6]
    - **‘re’**
  - >>> S1[1] = ‘*’
    - **TypeError: 'str' object does not support item assignment**
  - >>> S1 = ‘something else’
    - # This is ok



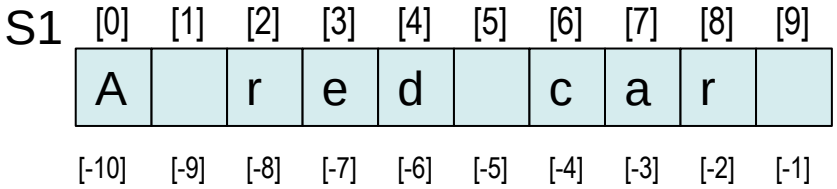
String Methods



- S1.count('r')
- **2**
- S1.split()
- **['A', 'red', 'car']**
- S1.replace('r', '*')
- **"A *ed ca* "**
- S1.upper()
- **"A RED CAR "**

NOTE: This is not an exhaustive list

String Methods



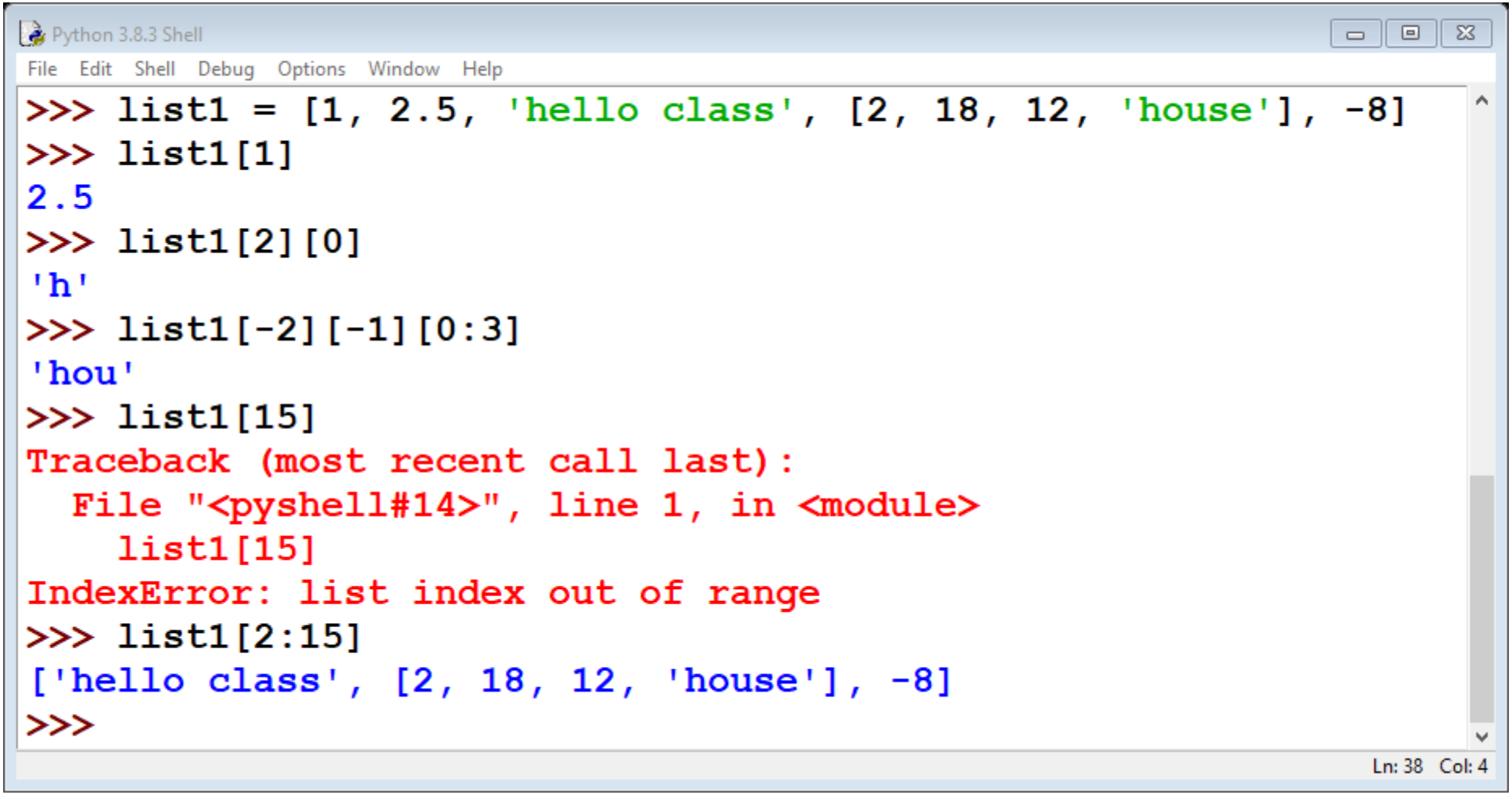
- S1.index('r')
- 2
- S1.index('x')
- ValueError
- S1.find('r')
- 2
- S1.find('x')
- -1
- S1.startswith('A red')
- True

NOTE: This is not an exhaustive list

Lists

- A collection data type that is ordered and changeable (mutable).
  - e.g. [1], [3, ‘hi’, 4.5, [1,2,3]], []
- Use [] to index items; similar to strings
- Can have multiple indices, e.g. list1[2][0], list1[-2][-1][-3]

Lists (Examples)



Your Turn …

- L1 = [1], [3, ‘hi’, 4.5, [1,2,3]]
  - >>> L1[1]
  - >>> len(L1)
  - >>> L1[3][0]
  - >>> L1[6]
  - >>> L1[-1]

range(n)

- *range(n)*: produces sequence 0, 1, 2, … *n*-1
- range([*i*,]*stop*[, *k*]): sequence starts at *i* (instead of 0) and incremented by *k* (instead of 1)
- range(5) → produces sequence 0, 1, 2, 3, 4
- range(3, 20, 4) → produces 3, 7, 11, 15, 19
- >>> L = [1, 5, 7, 2, 8]
- >>> for i in range(len(L)):

  L[i] = L[i]+10
- >>> L
  - [11, 15, 17, 12, 18]

List Methods

- L = [1,2,3,[4,5,'a','b'], 'hello', '6', 7]
  - L.append([8,9])
  - **[[1, 2, 3, [4, 5, 'a', 'b'], 'hello', '6', 7, [8, 9]]**
  - L += [8,9]
  - **[1, 2, 3, [4, 5, 'a', 'b'], 'hello', '6', 7, 8, 9]**
  - L.index(3)
  - **2**

NOTE: This is not an exhaustive list. We are always starting with initial value of L.

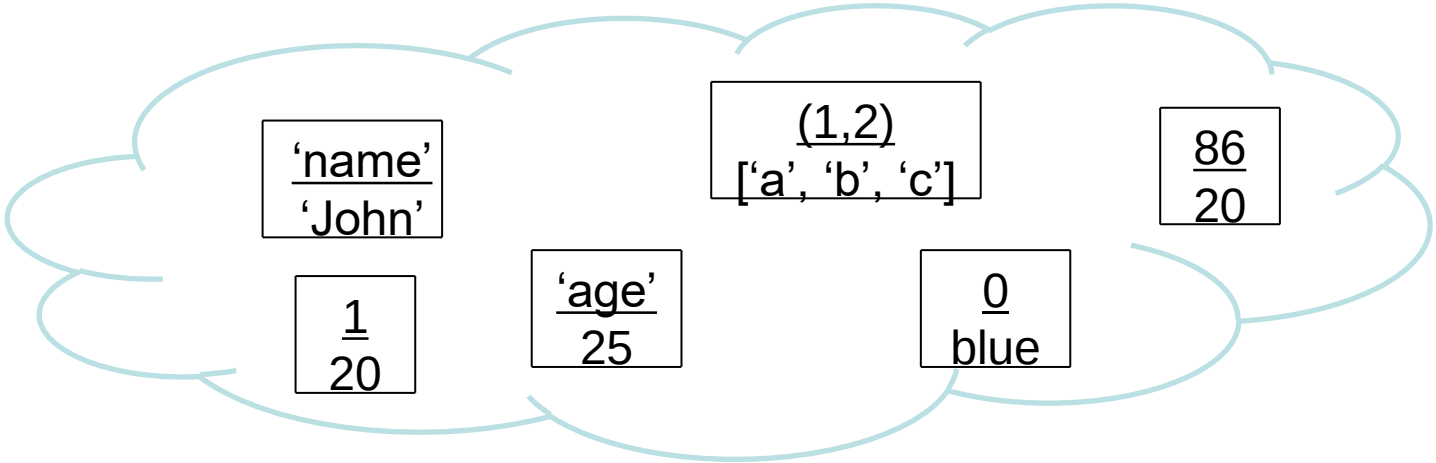
Dictionaries

- *dict*: an ***unordered*** collection of key-value pairs
  - ***mutable***
  - unordered → no notion of index positions
  - keys are unique → a key-value item ***replaces*** existing item with same key
  - duplicate values are fine

- d1 = {“name”: “John”, “age”: 25, (1,2):[‘a’, ‘b’, ‘c’], 0: “blue”, 86: 20, 1:20}

- d2 = dict([(“name”,“John”), (“age”, 25), ((1,2), [‘a’, ‘b’, ‘c’]), (0, “blue”), (86, 20)])

Dictionaries



- **d1 = {"name": "John", "age": 25, (1,2): ['a', 'b', 'c'], 0: "blue", 86: 20}**
- d1["name"]
- **"John"**
- d1[(1,2)]
- **['a', 'b', 'c']**
- d1["blue"]
- **KeyError**
- d1["blue"] = 1
- **# adds new key-value pair "blue":1**

Dictionaries

- **d1 = {'name' : 'John', 'age' : 25, (1,2):['a', 'b', 'c'], 0: 'blue', 86: 20, 1:20, 'blue':1}**
- d1[“blue”]
- 1
- del d1[‘age’]
- # deletes key-value pair “age”: 25
- d1[‘name’] = ‘Ty”
- # replaces key-value pair “name”: “John” with “name”: “Ty”
- d1[d1[“blue”]]
- 20

Membership Operator

- *in*: tests for membership, returns True or False
- *not in*: tests for non-membership, returns True or False
- L = [1, [2,3], “ab”, -23] S = “Python is great!”
  - [2,3] in L
  - **True**
  - 2 in L
  - **False**
  - 2 not in L
  - **True**
  - “on is” in S
  - **True**
  - “eat” not in S
  - **False**

Comparison Operators

- Basic comparison operators: <, <=, ==, !=, >=, >
  - a = 4, b=12
  - a < b → True
  - a == b → False
  - a >= b, a != b, a <= b → (False, True, True)
- Can be chained
  - 2 <= a < b <= 20 → True

Logical Operators

- 3 logical operators: and, or, not
  - and and or use short circuit logic → return operand that determined result, rather than a Boolean value (unless operands are Boolean)
  - Rules for expressions with non-Boolean types
    - int: 0 → False; All other values → True
    - float: 0.0 → False; All other values → True
    - list: [] (i.e. empty list) → False; All other values → True
    - str: '' (i.e. empty string) → False; All other values → True
  - 5 and 2
  - **2**
  - 0 and 2
  - **0**

Your Turn …

- Use short circuit evaluation to determine the results:
  - 5 and 2
  -
  - 0 and 2
  -
  - 0 or 7 or 8
  -
  - 6 or 6 or 9
  -
  - not 6
  -
  - not 0
  -
  - not ‘0’
  -

Summary

- Collection data types
  - Range() function
- Boolean operators
  - Membership
  - Comparison
  - Logical

References

- [1] Programming in Python 3 A complete introduction to the python language (2nd Ed) by Mark Summerfield. Addison Wesley 2010.