Functions and Dictionaries

HIT137 Software Now

Week 04

We extend our respect to Elders - past, present and emerging - and to all First Nations people.

Outline

- Understanding of:
  - Construct lists and access items in those lists
  - Use methods to manipulate lists
  - Perform traversals of lists to process items in the lists
  - Define simple functions that expect parameters and return values
  - Construct dictionaries and access entries in those dictionaries
  - Use methods to manipulate dictionaries
  - Decide whether a list or a dictionary is an appropriate data structure for a given application

Introduction

- A **list** allows the programmer to manipulate a sequence of data values of any types
- A **dictionary** organizes data values by association with other data values rather than by sequential position
- Lists and dictionaries provide powerful ways to organize data in useful and interesting applications

Lists

- List: Sequence of data values (**items** or **elements**)
- Some examples:
  - Shopping list for the grocery store
  - Guest list for a wedding
  - Recipe, which is a list of instructions
  - Text document, which is a list of lines
  - Words in a dictionary
- Each item in a list has a unique index that specifies its position (from 0 to length – 1)

List Literals and Basic Operators

- Some examples:
  - ['apples', 'oranges', 'cherries']
  - [[5, 9], [541, 78]]
- When an element is an expression, its value is included in the list:
- Lists of integers can be built using range!

```
import math
x = 2
[x, math.sqrt(x)]
```

Output

```
[2, 1.4142135623730951]
```

```
first = [1, 2, 3, 4, 5]
second = list(range(1, 6))
print(first)
print(second)
```

Output

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

List Literals and Basic Operators

```
first = [1, 2, 3, 4, 5]
second = list(range(1, 4))

second = second + [4, 5]

print(first)
print(second)

print(first == second)
```

Output

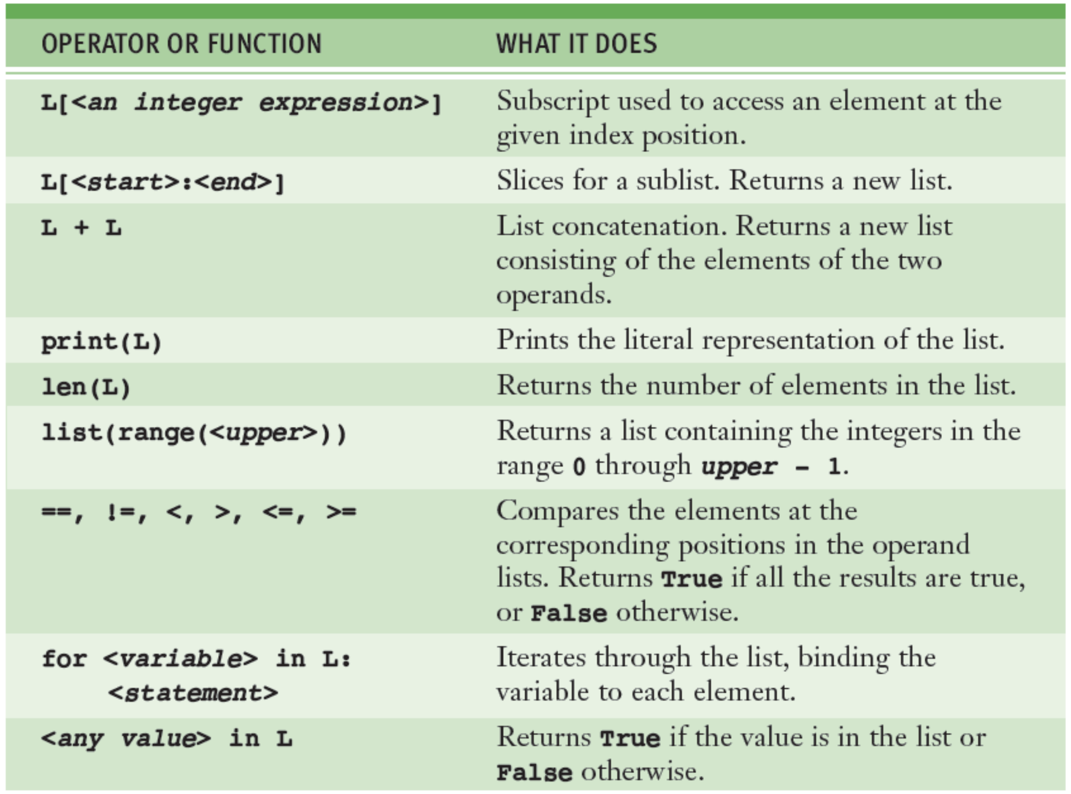
```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
True
```

```
print("1 2 3 4")
print([1, 2, 3, 4])
print(0 in [1, 2, 3, 4])
```

Output

```
1 2 3 4
[1, 2, 3, 4]
False
```

List Literals & Basic Operators



OPERATOR OR FUNCTION	WHAT IT DOES
`L[<an integer expression>]`	Subscript used to access an element at the given index position.
`L[<start>:<end>]`	Slices for a sublist. Returns a new list.
`L + L`	List concatenation. Returns a new list consisting of the elements of the two operands.
`print(L)`	Prints the literal representation of the list.
`len(L)`	Returns the number of elements in the list.
`list(range(<upper>))`	Returns a list containing the integers in the range `0` through `upper - 1`.
`==, !=, <, >, <=, >=`	Compares the elements at the corresponding positions in the operand lists. Returns `True` if all the results are true, or `False` otherwise.

`for <variable> in L:`
`    <statement>`
	Iterates through the list, binding the variable to each element.

`<any value> in L`	Returns `True` if the value is in the list or `False` otherwise.

[TABLE 5.1] Some operators and functions used with lists

Replacing an Element in a List

- A list is **mutable!**
  - Elements can be inserted, removed, or replaced
  - The list itself maintains its identity, but its **state**—its length and its contents—can change
- Subscript operator is used to replace an element:

```
example = [1, 2, 3, 4]
print(example)

example[3] = 0
print(example)
```

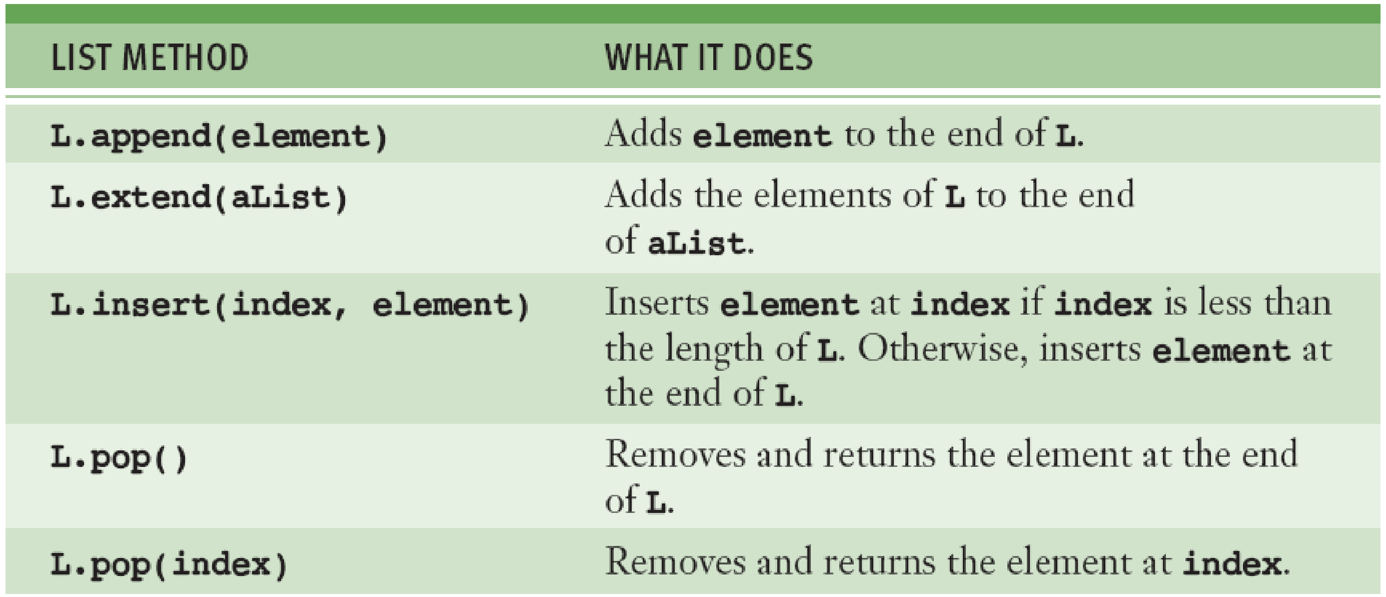
- Subscript is used to reference the target of the assignment, which is not the list but an element’s position within it

Coding

Refer to PDF_CODE

List Methods for Inserting and Removing Elements

- The list type includes several methods for inserting and removing elements



LIST METHOD	WHAT IT DOES
`L.append(element)`	Adds **`element`** to the end of **`L`**.
`L.extend(aList)`	Adds the elements of **`L`** to the end of **`aList`**.
`L.insert(index, element)`	Inserts **`element`** at **`index`** if **`index`** is less than the length of **`L`**. Otherwise, inserts **`element`** at the end of **`L`**.
`L.pop()`	Removes and returns the element at the end of **`L`**.
`L.pop(index)`	Removes and returns the element at **`index`**.

[TABLE 5.2] List methods for inserting and removing elements

List Methods for Inserting and Removing Elements

```
example = [1, 2]
print(example)

example.insert(1, 10)
print(example)

example.insert(3, 25)
print(example)
```

Output

```
[1, 2]
[1, 10, 2]
[1, 10, 2, 25]
```

```
example = [1, 2]
example.append(10)
print(example)

example.extend([11, 12, 13])
print(example)

print(example.pop())

print(example)
```

Output

```
[1, 2, 10]
[1, 2, 10, 11, 12, 13]
13
[1, 2, 10, 11, 12]
```

Searching and Sorting a List

Refer to PDF_CODE

Tuple

- A **tuple** resembles a list, but is immutable

```python
fruits = ("apple", "banana", "cherry")
print(fruits)

meats = ("chicken", "beef", "pork")
print(meats)

vegetables = ("carrot", "tomato", "cucumber")
print(vegetables)

food = fruits + meats + vegetables
print(food)
```

Output

```
('apple', 'banana', 'cherry')
('chicken', 'beef', 'pork')
('carrot', 'tomato', 'cucumber')
('apple', 'banana', 'cherry', 'chicken', 'beef', 'pork', 'carrot', 'tomato', 'cucumber')
```

- Most of the operators and functions used with lists can be used in a similar fashion with tuples

The Syntax of Simple Function Definitions

- Definition of a function consists of header and body

```python
def square(x):
  """Returns the square of x."""
  return x * x

print(square(3))
```

Output

```
9
```

- Syntax of a function definition:

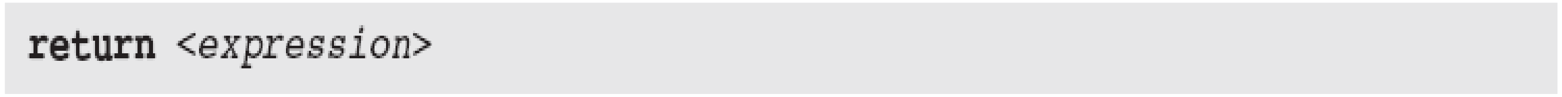
```
def <function name>(<parameter-1>, …, <parameter-n>):
    <body>
```

Parameters and Arguments

- A parameter is the name used in the function definition for an argument that is passed to the function when it is called
- For now, the number and positions of arguments of a function call should match the number and positions of the parameters in the definition
- Some functions expect no arguments
  - They are defined with no parameters

The Return Statement

- Place a return statement at each exit point of a function when function should explicitly return a value
- Syntax:



```
return <expression>
```

- If a function contains no **return** statement, Python transfers control to the caller after the last statement in the function’s body is executed
  - The special value ***None*** is automatically returned

Boolean Function

- A Boolean function usually tests its argument for the presence or absence of some property
- Returns True if property is present; False otherwise
- Example:

```python
def odd(x):
  """Returns True if x is odd or False otherwise"""
  if x % 2 == 0:
    return False
  else:
    return True

print(odd(3))
print(odd(6))
```

Output

```
True
False
```

Defining a main Function

- **main** serves as the entry point for a script
- Usually expects no arguments and returns no value
- Definition of main and other functions can appear in no particular order in the script
- As long as main is called at the end of the script

Defining a main Function

```python
def main():
    """The main function for the program."""
    number = float(input("Enter a number: "))
    result = square(number)
    print("The square of", number, "is", result)

def square(x):
    """Returns the square of x"""
    return x * x

main()
```

Output

```
Enter a number: 5
The square of 5.0 is 25.0
```

Running main

Refer to PDF_CODE

Dictionaries

- A dictionary organizes information by association, not position
- Example: When you use a dictionary to look up the definition of “mammal,” you don’t start at page 1; instead, you turn to the words beginning with “M”
- Data structures organized by association are also called tables or association lists
- In Python, a dictionary associates a set of keys with data values

Dictionary Literals

- A Python dictionary is written as a sequence of key/value pairs separated by commas
  - Pairs are sometimes called entries
  - Enclosed in curly braces ({ and })
  - A colon (:) separates a key and its value
- Examples:

```
phone_book = {'Sarah':'476-3321', 'Nathan':'351-7743'}
personal_info = {'Name':'Molly', 'Age':18, 'Gender':'Female'}
empty_dict = {}

print(phone_book)
print(personal_info)
print(empty_dict)
```

Dictionaries

Refer to PDF_CODE

Traversing a Dictionary – key and value

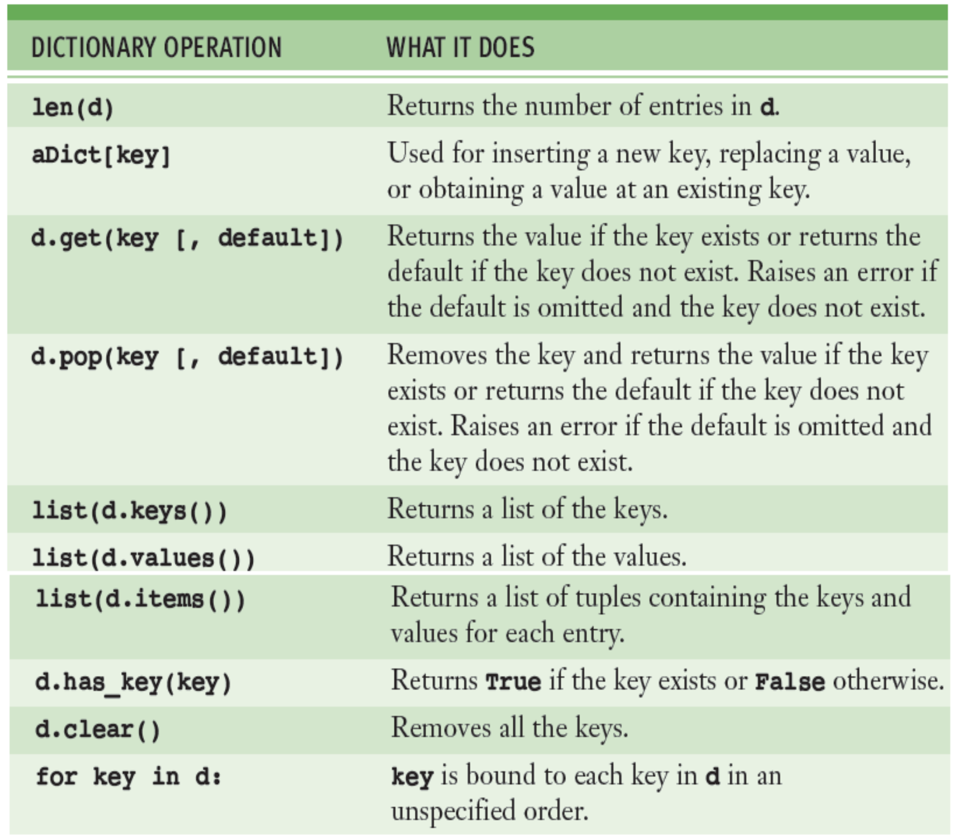
- To print all of the **keys** and their **values**:

```
grades = {90: 'A', 80: 'B', 70: 'C', 60: 'D', 50: 'E'}

grades.items()

for (key, value) in grades.items():
    print(key, value)
```

Traversing A Dictionary



DICTIONARY OPERATION	WHAT IT DOES
`len(d)`	Returns the number of entries in **d**.
`aDict[key]`	Used for inserting a new key, replacing a value, or obtaining a value at an existing key.
`d.get(key [, default])`	Returns the value if the key exists or returns the default if the key does not exist. Raises an error if the default is omitted and the key does not exist.
`d.pop(key [, default])`	Removes the key and returns the value if the key exists or returns the default if the key does not exist. Raises an error if the default is omitted and the key does not exist.
`list(d.keys())`	Returns a list of the keys.
`list(d.values())`	Returns a list of the values.
`list(d.items())`	Returns a list of tuples containing the keys and values for each entry.
`d.has_key(key)`	Returns **True** if the key exists or **False** otherwise.
`d.clear()`	Removes all the keys.
`for key in d:`	**key** is bound to each key in **d** in an unspecified order.

[TABLE 5.4] Some commonly used dictionary operations

The Hexadecimal System Revisited

```python
hexToBinaryTable = {
    '0': '0000', '1': '0001', '2': '0010', '3': '0011',
    '4': '0100', '5': '0101', '6': '0110', '7': '0111',
    '8': '1000', '9': '1001', 'A': '1010', 'B': '1011',
    'C': '1100', 'D': '1101', 'E': '1110', 'F': '1111'
}

def covert(number, table):
  """Builds and returns the base two representation of number."""
  binary = ''
  for digit in number:
    binary += table[digit]
  return binary

print(covert('1A', hexToBinaryTable))
```

Tutorial Session

Start
Solving
Tutorial
Questions

Resources

Some great resources to start your work :

- Fundamentals of Python: First Programs
- https://www.w3schools.com/python/default.asp
- https://python.swaroopch.com/