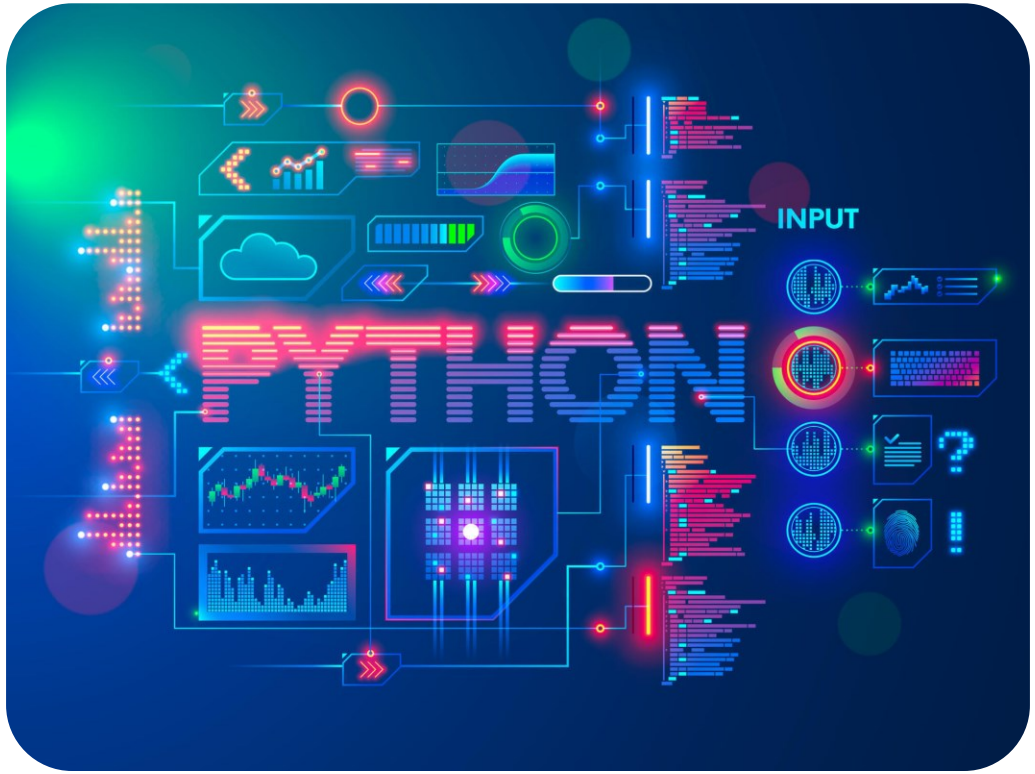
COLUMBUS STATE
COMMUNITY COLLEGE



WIIT 7740: Scripting with Python

Week 6: Dictionaries and String Manipulation, Exceptions

Dictionaries: Purpose

- Dictionaries are used to store data as key-value pairs
- Use a key to look up the corresponding value in a dictionary, sort of like names in a phone book
- Keys are ordered and do not allow duplicates
- Dictionaries are mutable so keys-value pairs can be added/removed/updated as needed
- A value that a key corresponds to in a dictionary can be changed
- Key-value pairs of a dictionary can be iterated over

Dictionaries: Examples

- Mapping of one set of values to another
- For each ***key*** in a dictionary, there is a corresponding ***value***

```
{}
```

```
{'alpha': 'A', 'bravo': 'B', 'charlie': 'C'}
```

```
{'x': True, 'y': False, 'z': False, 'w': True}
```

Accessing Values from Dictionaries

```
german = {'eins': 1, 'zwei': 2, 'drei': 3}

german['eins']   → 1

german['zwei']   → 2

german['drei']   → 3
```

Mutating Dictionaries

```
nato = {'A': 'alpha', 'B': 'bravo', 'C': 'charlie'}

nato['C'] = 'charlie'

nato['D'] = 'delta'
```

Dictionaries Key Types

The keys can be of any **immutable** type

```
{-3: 'integer', 27.004: 'rational', 'pi': 'irrational'}
```

Dictionaries have Length

The length of a dictionary is the number of key-value pairs

```
cipher = {'a': 17, 'b': 6, 'c': 22, 'd': 4, 'e': 10}

len(cipher)     → 5
```

Checking Dictionaries for Keys

The `in` operator evaluates whether dictionary has a given key

```
cipher = {'a': 17, 'b': 6, 'c': 22, 'd': 4, 'e': 10}

'a' in cipher        → True

'f' in cipher        → False
```

Dictionary Methods

Can get the keys and the values separately

```
nato = {'A': 'alpha', 'B': 'bravo', 'C': 'charlie'}

nato.keys()   →   ['A', 'B', 'C']

nato.values()   →   ['alfa', 'bravo', 'charlie']
```

Iterating a Dictionary

Using a **`for`** loop on a dictionary

```
for letter in nato:
    code_name = nato[letter]
    print(letter, "is called", code_name)
```

Output:

A is called alpha
B is called bravo
C is called charlie

Dictionary “items” Idiom

The **`items`** method is convenient in **`for`** loops

```
for letter, code_name in nato.items():
    print(letter, “is called”, code_name)
```

Output:

A is called alpha
B is called bravo
C is called charlie

Overview of Compound Data Types

Compound Data Type	Syntax	Conversion Function	Mutable?	Indexed by	Contents
String	””	str()	No	Integers starting at 0	Characters
Tuple	()	tuple()	No	Integers starting at 0	Anything
List	[]	list()	Yes	Integers starting at 0	Anything
Dictionary	{}	dict()	Yes	Any immutable data	Anything

Quick References: Dictionaries, Tuples

Dictionaries map keys to values:

`dict() → { }`	`{ key : value }`	`for k in d:`	`min(d) → key`
`len(d)`	`d.values()`	`for k, v in d.items():`	`key in d`

Tuples are essentially immutable lists:

`tuple() → ()`	`(0, 1, 200) < (0, 2, 3)`	`(a,)`
`len(x)`	`x, y = y, x`	`(a, b) + (c, d)`

Strings Manipulation

Key Points:

- Strings are immutable (you **cannot** change them “in place”)
- Strings have Methods
- With Strings, you can:
  - Identify length, min and max values, first and last values
  - Iterate over strings using loops
  - Traverse them using an optional 3rd argument to the slice which always starts with the first letter specified (or 1st letter if no specification)
    - Example: `'my_string'[0:6:2] == 'm_t'`
  - Traverse them backwards [::-1] and forwards [::1]
  - Find substrings within strings with the ‘in’ and ‘not’ operators
  - Combine Strings with + operator, or +=
  - Replicate Strings with the * operator

Exceptions

- Occur during execution of a program
- Cause the program to halt (“crash”)
- Error message shows ***type*** of exception and ***line number***

Examples of Exceptions

ZeroDivisionError: dividing a number by zero

IndexError: access an element of a list with an invalid index

NameError: using a variable that is not defined

See: https://docs.python.org/3/library/exceptions.html

Handling exceptions

You (the programmer) can decide:

- ***Where*** to detect an exception
- ***How*** to handle it

```
try:
    # where exception can occur
except TypeOfException:
    # how to handle that type of exception
```

Handling exceptions of different types

Always handle specific types of exceptions

```
try:
    # where exception can occur
except ZeroDivisionError:
    print("Oops, divided by zero.")
except NameError:
    print("Oops, undefined variable.")
except IndexError:
    print("Oops, invalid index.")
```

More topics on exceptions

The exception type hierarchy

How to access the exception object and raise exceptions

COLUMBUS STATE
COMMUNITY COLLEGE

Practice Coding:
Class Activity