# OMIS 30: Intro to Programming (with Python)

## Week 2, Class 2

Introduction to Programming
Instructor: Denis Vrdoljak

## Mutable vs Immutable

- Mutable = the item can be changed after created
- Immutable = the item cannot be changed after created
- All primitives are immutable

## Iterable Data Types in Python

- String
- Ways to hold and group the primitive data types include:
  - List
  - Dictionary
  - Tuple
  - Set

## String

- A string in Python is a sequence of characters
- It is a derived data type
- Can use “” or ‘’ to designate
- Strings are immutable

**Examples: “w”, “Dog”, “h7jb67”, ‘four’, ‘4’,**

**“The cat in the hat.”**

## String Methods and Operations

- str(), " " or ' '    - creates an empty string variable
- There is no char type, just a string with length 1
- Concatenation
  - “mystring” + “anotherstring” → “mystringanotherstring”
- “mystring” * 3 → 'mystringmystringmystring'
- """ """     - triple quotes, used for block comments on multiple lines
  - Example: """This is a long string that is
    split over two lines """

## More String Methods and Operations

- .upper(), .lower(), .capitalize(), .title()
  - mystring = ‘Abc dEf’
  - mystring**.upper()** = ‘ABC DEF’
    - Uppercase all letters
  - mystring**.lower()** = ‘abc def’
    - Lowercase all letters
  - mystring**.capitalize()** = ‘Abc def’
    - Capitalizes first letter of first word
  - mystring**.title()** = ‘Abc Def’
    - Capitalizes first letter of each word

## Advanced String Methods and Operations

- **str.strip()**
  - Used to remove ’unwanted’ characters from the start & end of a string:
    - mystr = "----Our string we want----"
    - mystr.strip(‘-’) = ‘Our string we want’
- **str.split()**
  - Used to split a string into multiple items on a character - returns a list of those strings
    - mystr = “eat, drink, sleep”
    - mystr.split(‘,’) = [‘eat’, ‘drink’, ‘sleep’]

## List

- Unlimited length, known order, mixed data types, mutable
    - y = [1,2,3]
    - mylist = [“the”, “cat”, “in”, “the”, “hat”]
    - another_list = [1, “the”, 3.45, True]

## List Methods and Operations

- x = list((1,2,3)) or x = [1,2,3]
  - Creates a list
  - To create a blank list is: x = list() or x = []
- mylist = [4,5,6]
- x + mylist -> [1,2,3,4,5,6] (adds both lists together but doesn't assign it to anything)
- x * 3 -> [1, 2, 3, 1, 2, 3, 1, 2, 3] (makes 3 copies of list but doesn't assign it to anything)

## More List Methods and Operations

- To start: x = [1,2,3]
- .append() - adds an item to the end of the list
  - x.append(4) -> [1,2,3,4]
- .remove() - removes an item from the list (from the end)
  - x.remove(4) -> [1,2,3]
- .pop() - pops an item of the end of the list and returns it
  - x.pop() -> 3
  - x.pop(0) -> 1 (pop(0) = front of the list)
- .extend() - extends the first list by adding the 2nd list to it
  - y = [9,10]
  - x.extend(y) -> [2,9,10]

## Advanced String and List Methods

- **str.join()**
  - Used to concatenate a sequence of strings into one string
  - .join() takes a list as argument
  - separator = “-”  # a string
  - sequence = (“join”, “me”, “together”)  # a list of strings
  - separator.join(sequence) = “join-me-together”
  - “ ”.join(sequence) = “join me together

## Dictionary (dict)

- A mutable unordered set of key:value pairs, with unique keys
  - Syntax: sound_dict = {“cat”: “meow”, “duck”: “quack”}
  - To access a given value, call the key:
    - >> sound_dict[“duck”]
    - Returns: “quack”
  - To assign a new key:value pair or reassign an existing key:
    - >> sound_dict[“cow”] = “moo”
    - >> print(sound_dict)
    - Returns: {“cat”: “meow”, “duck”: “quack”, “cow”: “moo”}

## Dictionary Methods and Operations

- sound_dict = {“cat”: “meow”, “duck”: “quack”, “cow”: “moo”}
- .keys() - returns a list of the keys of the dictionary
  - sound_dict.keys() = dict_keys(['cat', 'duck', 'cow'])
- .values() - returns a list of the values of the dictionary
  - sound_dict.values() = dict_values(['meow', 'quack', 'moo'])
- .items() - returns a list of tuples of (key,value)
  - sound_dict.items() = dict_items([('cat', 'meow'), ('duck', 'quack'), ('cow', 'moo')])

## Advanced Methods

- **zip()**
  - Used to pair up the elements of two lists (or other iterable) based on shared index
    - odd = (1,3,5), even = (2,4,6)
    - >> print(list(zip(odd, even)))

      [(1,2),(3,4),(5,6)]
  - Can also be used with dictionaries:
    - students = [“Matt”, “Jane”, “Bob”], grades =  [82, 97, 70]
    - >> print(dict(zip(names, grades)))

      {“Matt”:82, “Jane”:97, “Bob”:70}

## Tuple

- An immutable ordered list with a known number of elements.
  - Syntax: x = (1,4,6)
  - Immutability refers to the inability to be changed after the original assignment.
  - Tuples, like all the primitive data types, are immutable.

## Set

- An unordered collection of UNIQUE items.
  - Syntax: x = {4,1,6} or x = set((4,1,6))
    - If y = {4,4,6,1} -> y = {4,6,1} (the extra 4 is removed because its not unique item)
  - Cannot update an item only add or remove
    - set.add() -> adds that item to the set
      - x.add(7) -> {1, 4, 6, 7}
    - remove() -> removes that item from the set
      - x.remove(1) -> {6, 4, 7}

## Basic Built-ins

- Introducing a few useful/common built in functions:
  - len() <- returns the length of that object
  - type() <- returns the type of that object

## Indexing

- An iterable is any data type that can be used in a sequential fashion to find the next item, which includes string, list, tuple, dictionary, etc.
- We use the iterable property when searching through the various items to find a specific item, which is called indexing:
- >> mylist = [“the”, “cat”, “in”, “the”, “hat”]
- Pythno is ‘zero-based’ so indexing for the first item:
  - >> mylist[0]
  - Returns “the”

## More Indexing

- mylist = [“the”, “cat”, “in”, “the”, “hat”]
  - mylist[1]  -> “cat”
  - mylist[-1] -> “hat”
  - mylist[-4] -> “cat”
- mystr = ‘python’
  - mystr[0] = ?
  - mystr[-1] = ?

## Slicing

- To call up a subset/part of a list, we use a slice
- Slice syntax = [# to start with, # to end on (does not include): step]:
  - If either of the first two numbers are left blank - defaults to the start or end of the iterable
  - If the step is left blank - defaults to a step of 1
- Examples: mylist = [“the”, “cat”, “in”, “the”, “hat”]
  - mylist[0:2] returns [“the”, “cat”] (includes items 0 and 1, but not 2)
  - mylist[2:3] returns [“in”] (only include item 2, equivalent to indexing mylist[2])
  - mylist[2:] returns [“in”, “the”, “hat”] (the remainder of the list)
  - mylist[:-1] returns [“the”, “cat”, “in”, “the”] (everything up to the last item)

## More Slicing fun

- Examples: mylist = [“the”, “cat”, “in”, “the”, “hat”]
  - mylist[0:4:2] returns ['the', 'in'] (first item then step of 2)
  - mylist[::-1] returns ['hat', 'the', 'in', 'cat', 'the']  (reverses!)
  - mylist[4:8] returns [‘hat’]
- Example: mystr = ‘Python’
  - mystr[0:2] = ?
  - mystr[4:6].upper() = ?
  - mystr[1:5:3] = ?
  - mystr[::-1] = ?

## Variable Names

- Case matters (mystr is a different variable than MyStr)
- Cannot start with a number
- Usually variable names are in all lowercase
  - Can use underscores to make them more readable
    - E.g.: word_dict or my_list
- Keep variable names short (you might have to write them a lot!)
- ‘Counter’ variables are often a single letter like: i,j,k
- Try to name variables something that is easy to read for you and other programmers (i.e., avoid lowercase “L” as it looks like uppercase “I” and pipe: l, I, |)