COMP 10261: Python Review

Sam Scott, Mohawk College, September 2022.

WHAT IS THIS?

This is a quick primer on syntax, lists, and dictionaries in Python. Familiarity with Python is assumed so hopefully a lot of this is review. For more details, the free textbook *Think Python* is a good resource.

Warning: We are using Python 3, which is not backwards compatible with Python 2. But lots of people still use Python 2, so code you find on the Internet might not always work.

TYPES

Python is dynamically typed. No need to declare variables. Variables can store any type.

Important types include `int`, `float`, `str`, `bool`, `list` and `dict`.

Use the `type` function to find out the type of a value.

Convert between types using the type name function (e.g. `int("34")` returns `34`).

BASIC SYNTAX AND DOCUMENTATION

No semicolons. Code blocks are indented instead of being enclosed in braces { }. Use a **colon (:)** before a block. Use **`and`**, **`or`**, and **`not`** instead of `&&`, `||`, and `!`. Use **#** for a comment. Use **"""** (triple quotes) for a comment block or docstring. (Also, use **##** to get an underlined comment in Pyzo.)

See **documentation_standards.py** in the Resources module on Canvas.

SPECIAL OPERATORS

```
2 ** 10                    # means 2 to the power 10
"hi " + "there"            # string concatenation
"hi" * 5                   # evaluates to "hihihihihi"
```

INPUT AND OUTPUT

```
print(item1, item2, item3…, sep="", end="")
      # sep="" is optional. Otherwise spaces between the items.
      # end="" is optional. Otherwise carriage return at the end.

x = input(prompt)      # get keyboard input (prompt optional)

y = float( input() )  # get an input and convert to float
```

CONTROL FLOW

SELECTION (IF)

```
if boolean_expression:
      indented block of code
elif boolean_expression:          # optional "else if"
      indented block of code
else:                             # optional else
      indented block of code
```

REPETITION (WHILE)

```
while boolean_expression:
      indented block of code
```

REPETITION (FOR)

```
for variable in range(min, max, step):      # step and min are optional
      indented block of code
for variable in list                        # for-each
      indented block of code
```

FUNCTIONS

```
def function_name(parameter_list):     # no type declarations
      indented block of code
      return value                     # Optional – default is None

variable = function_name(argument_list)     # calls the function
```

FUNCTIONS WITH MULTIPLE RETURN VALUES

```
def function_name(parameter_list):
      indented block of code
      return value1, value2

var1, var2 = function_name(argument_list)
```

This is a Python **tuple**. A tuple is an immutable list. You can define a tuple with brackets: **x = (3,4,5)**

or without brackets: **x = 3,4,5**

DEFAULT VALUES AND NAMED (KEYWORD) ARGUMENTS

```
def foo(a,b=2,c=3):          # default values for b and c
      return a+b+c

x = foo(1)                   # uses defaults for b and c
x = foo(1,99)                # uses default for c only
x = foo(1,99,-4)             # uses no defaults

x = foo(1,c=-45)             # passes a by position, c by name
x = foo(c=1, b=2, a=3)       # specifies parameters by name
```

Python Lists

Lists are a lot like arrays, except that they can be expanded or contracted as necessary.

List Literals

Literals are values that are typed directly into a program (like `32` or `"hello, world"`). List literals are comma separated lists of values or expressions enclosed in square brackets.

`a = ["fda", 45, "23"]`	Creates a list and assigns it to `a`
`a = [b, c, d, b+c+d]`	Creates a list with values from variables `b`, `c` and `d`
`a = []`	Creates an empty list

Basic List Operators

Operators are symbols or keywords in the language that change or combine the contents of variables. Here are some important list operators:

`a + b`	Concatenates `a` and `b` into a new list
`a * n`	Creates a new list by repeating (`n` times) the contents of the list `a`
`a[i]`	Accesses element `i` of list `a`, ranging from `-len(a)` to `len(a)-1`
`e in a`	Returns `True` if `e` is an element of `a`
`e not in a`	Returns `True` if `e` is not an element of `a`
`del a[i]`	Removes the selected element from `a` (also works with slices)

The Slice Operator (:)

`a[i:j]`	Returns a copy of a slice of list `a` from `i` to `j`, not including `j`.
`a[:j]`	Returns a copy of a slice from `0` to `j`, not including `j`.
`a[i:]`	Returns a copy of a slice from `i` to the end of the list.
`a[i:j]=[1,2,3]`	Replaces a slice in list `a`.
`a[i:j]=[]`	Deletes a slice from list `a`.

List Functions

The following are built in global functions that operate on lists.

`len(a)`	Returns the length of `a`
`sum(a)`	Returns the sum of the elements in `a`
`min(a)`	Returns the minimum element in `a`
`max(a)`	Returns the maximum element in `a`
`sorted(a)`	Returns a new sorted list with the same contents as `a`
`sorted(a, reverse=True)`	Sorts in descending order

LIST METHODS

In Python, lists are objects with a number of built-in methods.

`a.append(e)`	Adds element `e` to the end of `a`
`a.extend(b)`	Adds all elements from list `b` to the end of `a`
`a.remove(e)`	Deletes the first element that is equal to `e` (using ==)
`a.insert(i, e)`	Inserts `e` into `a` at index `i`
`a.index(e)`	Returns the index of the first element equal to `e` (Raises a `ValueError` exception if `e` is not in the list)
`a.count(e)`	Returns a count of the number of times `e` appears in `a`

PROCESSING LISTS

To process a list is to "visit" (print, change, etc.) every element of the list.

In Python the `for` loop is actually a for-each loop. You can use it like a regular for loop by using the `range` function to create a numerical range

- `range(10)` creates the range `{0…9}`
- `range(5, 10)` creates the range `{5…9}`
- `range (1, 11, 2)` creates the range `{1,3,5,7,9}`
- The `reversed` function will reverse a range.

Use Loop 1 for visiting in order without changing. Loop 2 is for visiting and changing. Loop 3 is for visiting in reverse order.

1
```
for e in l
    visit e
```

2
```
for i in range(len(l))
    visit l[i]
```

3
```
for i in reversed(range(len(i)))
    visit l[i]
```

ASIDE: A NOTE ON STRINGS

Strings are basically immutable (i.e. unchangeable) lists of characters. Because of this, many of the operators, functions and methods reviewed here work the same way on strings as they do on lists. The only difference is that no operation is allowed that would mutate (change) the contents of a string.

We will review Python strings in more detail next week.

Python Dictionaries (and Sets)

A **dictionary (the dict type in Python)** is like an **associative array** in JavaScript and PHP or a **HashMap** in Java. Dictionaries store keys using **hashing**, which you will learn about in the **Data Structures** course.

Dictionary Creation

```
d = {"name":"Alex", "age":45, "authorized":True}
```

`d = {}` ← empty dictionary `d = dict()` ← empty dictionary

Dictionary Operators and Methods

Operators are symbols or keywords in the language that change or combine the contents of variables. Here are some important list operators:

`d[key]=value`	Stores a `value` for a `key`
`d[key]`	Accesses the value paired with `key` or throws `KeyError`
`d.get(key, v=None)`	Returns the value associated with `key`, or `v` if `key` not found
`key in d`	Returns `True` if `key` is an element of `d`
`del d[key]`	Removes the selected `key` from `d` or throws `KeyError`
`d.clear()`	Clears out `d`

An Aside on Catching Exceptions

Exception handling in Python works much the same way as it does in Java, C#, etc.

```
try:
    print(d["mykey"])
except KeyError:                ← leave out the type to catch all exceptions
    print("key not found")
```

Sets

A set is a **dict** that only stores keys. You can add and remove keys ask whether a key is in the set.

`s = set()` `s.add(key)` `s.remove(key)` `key in s`

Processing Dicts and Sets

To process a dictionary (or set) is to "visit" every key in the dictionary (or set).

```
for key in d:
    print(d[key])
```

Dict and Set Functions

The functions `len`, `sum`, `min`, `max`, and `sorted` can be passed `dict` and `set` arguments. They perform their operations on the keys, not the values.

Exercises

Use these optional exercises to brush up on Python and try out lists, slices, dictionaries, and sets.

CONTROL STRUCTURES AND FUNCTIONS

1. Write a function that chooses a random number from 1 to 10 and then plays a guessing game with the user, saying "higher" or "lower" until they get it right. Make it robust.

```
import random
x = random.randint(1,10)
```

LISTS

2. Write a function called `createList(value, length)` that returns a list with a given initial `value` and `length`.

    a. Can you create and return the list on a single line?
    b. What happens when the user passes a list to the value parameter?
    c. Can you create a new version of `createList` that will work like this instead?

```
>>>createList([1,2,3], 10)
[1,2,3,1,2,3,1,2,3,1]
```

3. Write a function that takes a list argument and returns a tuple of two items selected at random. Write a line of code that calls the function and stores the results in variables.

4. Write a function called `fixLength` that takes a list and an integer `n` as parameters. Return a copy of the list that is exactly `n` elements long. Do this by either cutting of the tail of the list or padding the tail with the value 0. (Hint: Use the **slice** and ***** operators.)

    a. Add an optional third parameter. If the user passes no value or passes the value `True`, pad the end of the list, otherwise pad the front.
    b. Make the function robust against bad parameter values and test it thoroughly.

DICTIONARIES & SETS

5. Write a function to count the number of unique words in a given string. Use **s.split()** to split a string into a list of words, then put the words into a set. Use **s.lower()** to convert a string to lower case.

6. Write a function to count the number of times each word occurs in a given string. Use **s.split()** to split a string into a list of words, then use a **dict** to count the occurrences of each word. Use **s.lower()** to convert a string to lower case.