CS 146 Fall 2025 – Midterm 2 "Cheat Sheet"

Numeric Operators

+, -, /, *, **: Addition, subtraction, true division, multiplication, power
//: Floor division: Round division result down to nearest whole number
%: Modulo: Evaluate to remainder of division

Comparison Operators

==, !=: Equals, not equals
>, >=, <, <=: Greater than, greater than or equals, less than, less than or equals

Boolean Operators

not op, op1 and op2, op1 or op2: Logical NOT of op, AND of op1 and op2, OR of op1 and op2

Indexing Operator and slicing

seq[idx]: Get or assign item of `seq` at index `idx` or get or assign value associated with key `idx` in dictionary
seq[start:stop(:step)]: Copy of subsequence of `seq` from inclusive `start` to exclusive `stop` by `step`
seq[:]: Copy all items in `seq`

Precedence:

parentheses > indexing > ** > negate > *,/,//,% > +,- > comparisons > not > and > or

Range

range(stop): Equivalent to range(0, stop, 1)
range(start, stop[, step]): Create sequence of integers from inclusive **start** to exclusive **stop** by **step**

Input

- Reading input from the user
  input(message): Displays message to the user and returns what the user typed as a string
- Reading from a file with a for loop

```
with open(filename, "r") as file:
    for line in file:
        # do something with line (a string)
```

- Writing to a file
  open(filename, "w"): Write to file (overwrite any existing content)
  open(filename, "a"): Append to the end of existing contents
  file.write(item): Writes item to file (e.g. string, number) w/o trailing newline

Command Line Arguments

- Access command line arguments when file is run as a program

```
import sys
# __name__ is automatically set to "__main__" when file is run
if __name__ == "__main__":
    # For python program.py arg1 arg2, sys.argv is ["program.py","arg1","arg2"]
```

Built-in functions

abs(a): Return absolute value of number a

Strings

- The following functions are built-in
  len(string): Returns the number of characters in the string
  int(string), float(string): Converts numeric string to int or float
  str(object): Converts object, e.g. int or float to a string
  sorted(string): Returns the characters of the string as a list in sorted order

- String object methods
  `count(some_string):` Return number of occurrences of **`some_string`** in the string
  `index(some_string):` Returns the index of the first occurrence of **`some_string`** or error if it does not occur
  `upper(), lower(), capitalize():` Returns a new upper or lower-cased, or 1st letter upper-cased string
  `find(some_string):` Returns the first index that **`some_string`** occurs at in the string or -1 if not found
  `find(some_string, index):` Same as above, but starts searching at index
  `replace(old, new):` Return a copy of the string with all occurrences of old substituted with new
  `startswith(prefix):` Returns **`True`** if the string starts with prefix, False otherwise
  `endswith(suffix):` Returns **`True`** if the string ends with suffix, False otherwise
  `strip():` Returns a copy of the string with only the leading and trailing whitespace removed
  `split():` Return a list of the words in the string using whitespace as the delimiter
  `isalpha():` Return **`True`** if all characters in string are alphabetical and the string has at least one character
- String operators
  `string1 + string2:` Returns a new string that is the concatenation of string1 and string2
  `string * int:` Returns a new string that is string repeated int times
  `substr in string:` Returns True if substr is a substring of string, False otherwise

Lists

- Creating new lists
  `[]` creates empty list
  `[object1, object2, ...]` creates list containing objects
  `list(iterable)` creates a list from any iterable object (e.g., range, string)
- The following functions are built-in
  `len(list):` Returns the number of elements in `list`
  `sum(list), min(list), max(list):` Returns the sum, min, or max of elements in `list`
  `sorted(list):` Returns a new copy of the list in sorted order
- List object methods
  `count(item):` Returns the number if occurrence of `item` in the list
  `index(item):` Returns the index of the first occurrence of `item` in the list or error if it does not occur
  `append(x):` Adds `x` to the end of the list
  `extend(other_list):` Adds all elements of `other_list` the end of the list
  `insert(index, x):` Insert `x` before `index` in the list
  `pop():` Removes the item at the end of the list and returns it
  `pop(index):` Removes item at `index` from the list and returns it
  `remove(value):` Remove first occurrence of `value` from list
  `reverse():` Reverses the elements in the list in place
  `sort():` Sorts the elements of the list in place, returns None
- List operators
  `list1 + list2:` Returns a new list that contains the elements of `list1` followed by the elements of `list2`
  `list * int:` Returns a new list that contains the items in `list` repeated `int` times
  `item in list:` Returns True if `item` is an element of `list`, False otherwise

Sets

- Creating new sets
  `set()` creates empty set
  `{elt1, elt2, ...}` creates a new set with the given elements
  `set(iterable)` creates a set from any iterable object (e.g., string, list)
- The following functions are built-in and answer questions about sets
  `len(set):` Returns the number of elements in the set

- Set object methods
  `add(elt):` Adds `elt` to the set
  `clear():` Removes all elements from the set
  `pop():` Removes an arbitrary element from the set and returns it
  `remove(elt):` Removes `elt` from the set
  `union(set2):` Returns new set with union of itself and set2
  `update(set2):` Update itself with union of itself and set2
- Set operators
  `elt in set:` Returns True if `elt` is an element of `set`, False otherwise
  `set1 < set2:` Returns True if set1 is a proper subset of set2 (every element of set1 is in set2 and set1 != set2)
  `set1 | set2:` Returns union of the two sets (new set with elements from both set)
  `set1 & set2:` Returns intersection of the two sets (new set with only elements common to both sets)
  `set1 - set2:` Returns set difference (new set with elements `set1` not in `set2`)
  `set1 ^ set2:` Returns set symmetric difference (new set with elements in `set1` or `set2` but not both)

Dictionaries

- Creating new dictionaries
  `{}` creates empty dictionary
  `{key1:value1, key2:value2, ...}` creates a new dictionary with key-value pairs
- The following functions are built-in and answer questions about dictionaries
  `len(dict):` Returns the number of entries (key-value pairs) in the dictionary
- Dictionary object methods
  `clear():` Removes all entries from the dictionary
  `keys():` Returns an iterable object of all the keys in the dictionary
  `values():` Returns an iterable object of all the values in the dictionary
  `items():` Returns an iterable object of all (key, value) tuples in the dictionary
  `get(key[, item])`: Returns value associated with **`key`** if in dictionary, **`item`** otherwise. **`item`** defaults to None.
- Dictionary operators
  `item in dict:` Returns True if `item` is in the keys of `dict`, False otherwise

Tuples

- Creating new tuples
  `()` creates empty tuple
  `(object1, object2, ...)` creates tuple containing objects
- The following functions are built-in and answer questions about tuples
  `len(tuple):` Returns the number of elements in the tuple
- Tuple operators
  `item in tuple:` Returns True if `item` is contained in `tuple`, False otherwise
  `tuple1 + tuple2:` Returns a new tuple that is the concatenation of `tuple1` and `tuple2`

Classes

- Define a class **`DerivedClass`** that inherits/derives from **`BaseClass`**

```
class DerivedClass(BaseClass):
    def __init__(self, x):
        # Initialize instance variables, e.g.
        self.x_coord = x

    def a_method(self, y):
        # …
```

- Create an instance of a class: **`DerivedClass(4)`**

- **`print`** uses the **`__str__`** method
- Operators +, -, *, /map to methods **`__add__, __sub__, __mul__, __truediv__`**
- Operators ==, !=, <, <=, >, >= map to methods **`__eq__, __ne__, __lt__, __le__, __gt__, __ge__`**

Modules

- **`turtle`** module

  `forward(dist), backward(dist):` Move the turtle forward/backward by the length `dist`. Doesn't change heading.

  `right(angle) left(angle):` Turn the turtle right/left by `angle` (in degrees)

  `goto(x, y):` Move turtle to position `x, y`

  `setheading(angle):` Set the turtles heading to angle

  `circle(radius):` Draw a circle with specified `radius`; the center is `radius` above the starting position

  `dot(size):` Draw a filled circle with diameter `size` centered on current position of the turtle

  `penup():` Pull the pen up – no drawing when moving

  `pendown():` Put the pen down – drawing when moving

  `fillcolor(color):` Change the fill color to `color`, where `color` is a string

  `begin_fill(), end_fill():` Start and end filling shapes with fill color

- **`random`** module

  `randint(a, b):` Return a random integer `N` such that `a ≤ N ≤ b`

  `uniform(a, b):` Return a random floating point number `N` such that `a ≤ N ≤ b`

- **`math`** module

  `sqrt(num):` Return the square root of `num`

  `pow(x, y):` Return `x` to the power of `y` (i.e. `x ** y`)