CSE 160 23au Final Exam Cheat Sheet

```
# if/elif/else syntax
if condition1:
    # statements
elif condition2:
    # other statements
else:
    # more statements
```

```
# for loop syntax
for i in sequence:
    # statements

# function definition syntax
def function_name(param1, param2, ...)
    # function body
```

Function	Description
`range([`*start*`,] stop [, `*step*`])`	Returns a sequence of numbers from *start* (inclusive) to *stop* (exclusive) incremented by *step*
`len(`*lst*`)`	Returns the number of elements in *lst*

Lists

Function	Description
`lst = []`	Creates an empty list
`lst[`*idx*`]`	Returns the element in `lst` at index *idx*
`lst[`*start*`:`*end*`]`	Returns a sublist of `lst` from index *start* (inclusive) to index *end* (exclusive)
`lst[`*start*`:`*end*`:`*step*`]`	Returns a sublist of `lst` from index *start* (inclusive) to index *end* (exclusive), incrementing by *step*
`lst.append(`*elmt*`)`	Adds the element *elmt* to the end of `lst`. Returns `None`
`lst.extend(`*L*`)`	Extends `lst` by appending all elements in list *L* to end of `lst`. Returns `None`
`lst.insert(`*idx*`, `*elmt*`)`	Inserts an element *elmt* in `lst` at index *idx*. Returns `None`
`lst.index(`*elmt*`)`	Returns index of the first occurrence of *elmt* in `lst`, `Error` if *elmt* is not in `lst`
`lst.count(elmt)`	Returns the number of times *elmt* occurs in `lst`
`lst.remove(`*elmt*`)`	Removes first occurrence of *elmt* from `lst`, `Error` if *elmt* is not in lst. Returns `None`
`lst.pop([`*idx*`])`	Removes and returns the element at index *idx* (or last element if no parameter is specified) in `lst`

File I/O

Function	Description
`my_file = open(file_path)`	Opens file at given `file_path` for reading
`my_file = open(file_path, "w")`	Opens file at given `file_path` for writing
`my_file.close()`	Closes file `my_file`

```
# Process one line at a time
for line of text in my_file:
    # process line_of_text
```

```
# Process entire file at once
all_data_as_big_string = my_file.read()
```

Dictionaries

Function	Description
`my_dict = {}`	Creates a new dictionary
`my_dict[`*`key`*`]`	Returns the value associated with the given *`key`* in `my_dict`
`my_dict.keys()`	Returns sequence of keys in `my_dict`
`my_dict.values()`	Returns sequence of values in `my_dict`

Sorting

Function	Description

`sorted(`*`collection`*
`[,key=`*`sort_key`*`,`
`reverse=`*`bool_val`*`])`
	Returns a sorted copy of *`collection`*, based on optional sort key *`sort_key`* and optional sort preference *`reverse`*

`lst.sort([key=`*`sort_key`*`,` `reverse=`*`bool_val`*`])`	Sorts `lst` in-place, based on optional sort key *`sort_key`* and optional sort preference *`reverse`*. Returns `None`

Common Error Names

- `IndexError` - Index out of range
- `KeyError` - Key not found in dictionary
- `IndentationError` - Invalid indentation
- `TypeError` - Operation applied to invalid combination of types
- `ValueError` - Function gets properly typed argument, but invalid value
- `SyntaxError` - Invalid Python syntax
- `NameError` - Variable name not found
- `FloatingPointError` - Floating point operation fails
- `RuntimeError` - Otherwise unknown error

Graphs

Function	Description
`import networkx as nx`	Imports graph library and aliases the library name to `nx`, usable as `nx.Graph()`
`g = nx.Graph()`	Creates a new graph
`g.add_edge("A", "B")`	Adds an edge between nodes `"A"` and `"B"`, creating the nodes if they did not exist
`g.add_node("A")`	Adds node `"A"` to the graph
`g.neighbors("A")`	Returns a collection of the neighbors of the node `"A"`
`g.nodes()`	Returns a collection of nodes in the graph
`g.edges()`	Returns a collection of edges in the graph

Sets

Function	Description
`s1 = set()`	Creates a new empty set
`s1 = set([...])`	Creates a new set containing all of the elements within the given list
`s1 | s2`	Evaluates to union of `s1` and `s2`
`s1 & s2`	Evaluates to intersection of `s1` and `s2`
`s1 - s2`	Evaluates to difference of `s1` and `s2`

Classes

Function	Description
`__init__(self)`	Name of class constructor
`self`	Required parameter for all class methods (functions). Refers to specific instance of the class. Can hold any number of variables, known as **fields**.