

# 1 Functions

## 1.1 Function Definitions

Function def without parameters:

```
def foo():  
    #Function body is indented
```

Function def parameters:

```
def foo(param_1, param_2, ..., param_n):  
    #Function body is indented
```

## 1.2 Return vs Print

1. `print()` is not the same as `return`
  - (a) `print()` → Prints to the monitor. It does not give you a value you can work with.
  - (b) `return` → Isn't used to print things to the screen. It's used to give data back after a function has finished doing stuff.
2. A function only returns **once**. No matter how many return statements you put in the program the only one that matters is the first one that's reached.

# 2 Slicing

- Can be used on either lists, tuples, or strings.
- `x[start:stop: interval]`
- Like range, start is inclusive stop is exclusive.
- Interval default is 1
- Interval is optional

# 3 List

## 3.1 List Methods

Use `help(list.<method name>)` for information on a given method:

- `L.append(elem)` → Add element to the end of L and returns None.
- `L.extend(lst)` → Add all elements of lst to the end of L and returns None.

## Topics Review Sheet

---

- `L.insert(index, elem)` → Insert element at index of L pushing other elements forward and returns `None`.
- `L.pop()` → Remove and return the element at the end of L and returns the value that was removed.
- `L.pop(index)` → Remove and return the element at index of L and returns the value that was removed.
- `L.remove(elem)` → Remove first occurrence of element from L and returns `None`.
- `L.sort(elem)` → Sort the elements of L *in-place* and returns `None`.

### 3.2 Creating vs Modifying Lists

#### Creates new list:

- `z = x.copy()`
- `z = x[:]`
- `z = x + y`
- `z = sorted(x)`
- `z = reversed(x)`

#### Modifies a list in-place:

- `x.sort()`
- `x.append(num)`
- `x.remove(num)`
- `x.extend([num1, num2, ...])`
- `x.pop(index)`
- `x.insert(num)`

#### 3.2.1 List Comprehension

---

```
# Basic list comprehension
[expression for value in sequence]
# Conditional list comprehension with if
[expression for value in sequence if condition]
# Conditional list comprehension with if-else
[expression if condition else <default> for value in sequence]
```

---

## 4 Strings

### 4.1 String Functions

Recall, you can use `help(str.<method>)` to retrieve the docstrings associated with any of the below functions:

## Topics Review Sheet

---

- `x = str()` → Creates an empty string or converts an item to a string if given a parameter.
- `x = ""` → Defines an empty string and associated with the variable x.
- `x.replace(old, new)` → Returns a copy of the string with all occurrences of the substring old replaced by the string new. The old and new arguments may be string variables or string literals.
- `x.replace(old, new, count)` → Same as above, except only replaces the first count occurrences of old.
- `x.find(y)` → Returns the index of the first occurrence of item y in the string, else returns -1. y may be a string variable or string literal. Recall that in a string, the index of the first character is 0, not 1.
- `x.find(y, start)` → Same as `find(y)`, but begins the search at index start.
- `x.find(y, start, end)` → Same as `find(y, start)`, but stops the search at index end - 1
- `x.rfind(y)` → Same as `find(y)` but searches the string in reverse, returning the last occurrence in the string.
- `x.count(y)` → Returns the number of times y occurs in the string.

### Boolean String Functions:

- `x.isalnum()` → Returns True if all characters in the string are lowercase or uppercase letters, or the numbers 0-9.
- `x.isdigit()` → Returns True if all characters are the numbers 0-9.
- `x.islower()` → Returns True if all cased characters are lowercase letters.
- `x.isupper()` → Return True if all cased characters are uppercase letters.
- `x.isspace()` → Return True if all characters are whitespace.
- `x.startswith(y)` → Return True if the string starts a substring y.
- `x.endswith(y)` → Return True if the string ends with a substring y.

### Copy of a new string with modification:

- `z = x.capitalize()` → Returns a copy of the string with the first character capitalized and the rest lowercased.
- `z = x.lower()` → Returns a copy of the string with all characters lowercased.

## Topics Review Sheet

---

- `z = x.upper()` → Returns a copy of the string with all characters uppercased.
- `z = x.strip()` → Returns a copy of the string with leading and trailing whitespace removed.
- `z = x.title()` → Returns a copy of the string as a title, with first letters of words capitalized.

### 4.2 Join and Split

- `"<sep>".join(list)` → Returns a string where all elements in list have are separated by the separator in the string before join.
- `string.split("<sep>")` → Returns a list of substrings by splitting up the string based on the separating character.

### 4.3 Join and Split: Common Pattern

The generic pattern:

---

```
my_list = input_data.split(<separator>)
... data processing to build new list or modify my_list ...
outputstring "<separator>".join(my_list)
```

---

## 5 Dictionaries

Recall, you can use `help(dict.<method>)` to retrieve the docstrings associated with any of the below functions:

- `x = dict()` → Defines an empty dictionary and associated with the variable x.
- `x = {}` → Defines an empty dictionary associated with the variable x.
- `x[key] = value` → Given a dictionary x with will replace the value associated with the key or, if there is no key currently in x, it will create a new key value pair.
- `x.update(y)` → Given a dictionary x and y, update will all all key value pairs from y to x. This function is in place and therefore returns None.
- `x.items()` → An object which can be iterated over in order to retrieve tuples of the key-value pairs in a dictionary x.
- `x.keys()` → An object which can be iterated over in order to retrieve the keys in the dictionary x.

## Topics Review Sheet

---

- `x.values()` → An object which can be iterated over in order to retrieve the values in the dictionary `x`.

## 6 Iterating over Dictionaries

When iterating over a dictionary (`x`) we use the following form:

---

```
for key in x:  
    # Body of the loop
```

---

The former, as indicated by the looping variables name, only iterates over the keys in the dictionary. If you wish to iterate over both keys and values at the same time you can use the following:

---

```
for key in x:  
    # Body of the loop
```

---

## 7 Sets

### 7.1 Set Functions

1. `a = set()` → Creates an empty set and stores it in `a`.
2. `a.add(element)` → Adds a single element to a set.
3. `a.update(b)` → Adds all the elements from `b` to `a`. This function does not return anything.
4. `c = a.union(b)` → Creates a new set containing all of the elements from `a` and `b` and stores it in `c`.
5. `c = a.intersection(b)` → Creates a new set containing the intersection of `a` and `b` and stores it in `c`.
6. `c = a.difference(b)` → Creates a new set containing the intersection of `a` and `b` and stores it in `c`.

## 8 Conditionals

### 8.1 Conditional Branching

If Statement:	If-Else Statement:	If-Elif-Else Statement:
<hr/> <pre>if &lt;cond&gt;:     ...</pre> <hr/>	<hr/> <pre>if &lt;cond&gt;:     ... else:     ...</pre> <hr/>	<hr/> <pre>if &lt;cond&gt;:     ... elif &lt;cond&gt;:     ... elif &lt;cond&gt;:     ... else:     ...</pre> <hr/>

### 8.2 Short Circuiting

---

```
True or anything() # This is True
False and anything() # This is False
```

---

- Python won't evaluate the anything() part.
- You can use this to prevent errors from occurring in your code or having to next if statements:

---

```
if (len(my_str) > 10) and (my_str[10] == 'a'):
    print("the_tenth_character_of_my_string_is_", my_str[10])
```

---

### 8.3 Truthy and Falsy

Most all values in Python are treated as True with the exception of the following: (1) None (2) False (3) 0 (4) 0.0 (5) Decimal(0) (6) Fraction(0, 1) (7) [] (8) {} (9) () (10) '' (11) set() (12) range(0)

### 8.4 or Operator: A common error

1. Do not use the following method to check if x is equal to either 3 or 4 `x == 3 or 4`. It will always evaluate to the right expression, 4, which has the truthiness of true. This means the boolean expression will always evaluate to true.
2. Alternatives:

## Topics Review Sheet

---

(a) `x == 3 or x == 4`

(b) `x in [3, 4]`

3. Types of operators:

(a) **Binary operators (two operands):** and, or

(b) **Unary Operators (one operand):** not

## 9 Loops in General

### 9.1 For Loops: Iterating over a collection

---

```
x = <some collection>
for item in x:
    # body of the for loop
```

---

```
x = <some collection>
for i, item in enumerate(x):
    # i is the index of item
```

---

### 9.2 While Loops: If Statements that Just Keep Going

Key differences between for loops and while loops:

1. For loops iterate over collection. While loops iterate while boolean expression is true.
  2. For loops terminate when they run out of values. While loops terminate when a boolean expression is false.
- 

```
while <cond>:
    # Function body code below
    ...
```

---

### 9.3 For vs While Loop: Comparison Example

These two pieces of code are equivalent:

---

```
x = [1, 2, 3, 4]
for item in x:
    print(item)
```

---

---

```
x = [1, 2, 3, 4]
i = 0
while i < len(x):
    item = x[i]
    print(item)
    i += 1
```

---

### 9.4 Break vs Return vs Continue

- **continue** → Skips everything below it and goes back to the beginning of the loop to which it belongs.
- **break** → Exits loop it is apart of.
- **return** → Leaves function with return value.

### 9.5 Range Function Variations

The following are all variations of the range function:

1. `range(end)`
2. `range(start, end)`
3. `range(start, end, increment)`

**Note:**  $start \leq x < end$  for all  $x$  in the range.

## 10 Files

### 10.1 Reading from Files

Method 1:

---

```
file_object = open('filename')
lines = file_object.readlines()
for line in lines:
    print(line)
file_object.close()
```

---

Method 2:

---

```
with open('filename') as inf:
    lines = inf.readlines()
    for line in lines:
        print(line)
#automatic file close
```

---



## 10.2 Writing to Files

Method 1:

---

```
file_object = open('filename', 'w')
file_object.write('thing_to_write')
file_object.close() #automatic at program end
file_object.flush() #optional
```

---

Method 2:

---

```
with open('filename', 'w') as outf:
    outf.write('thing_to_write')
#automatic file close
```

---

## 11 Patterns

### Counting Pattern

---

```
def count(collection):
    counter = 0
    for item in collection:
        if <item meets condition>:
            counter += 1
    return counter
```

---

### Computing a Sum/Total

---

```
def sum(collection):
    total = 0
    for item in collection:
        total += item

    return total
```

---

### Finding (single thing) in a Collection

---

```
def find_thing(collection):
    for thing in collection:
        if <thing meets condition>:
            return thing
```

---

---

```
def find_thing(collection):
    found = None
    for thing in collection:
        if <thing meets condition>:
            found = thing
            break
    return found
```

---

### Filtering a collection

---

```
def filter(collection):
    new_list = []
    for thing in collection:

        if <thing meets criteria>:
            newlist.append(thing)

    return new_list
```

---

## Topics Review Sheet

---

### Finding best in collection

---

```
def find_best(collection):
    currentbest = ??
    for thing in collection:
        if <thing is better than current best>:
            currentbest = thing
    return currentbest
```

---

- If we're searching over a list and we want to return the largest or smaller number:  
currentbest = stufflist [0]
- If we're searching over a list of strings and we want to return the longest string:  
currentbest = stufflist [0] or currentbest = ""
- If you know the list contains only non-negative integers: currentbest = -1

## 12 File Patterns

Usual Sum/Total:

---

```
def foo(some_list):
    total = 0
    for item in some_list:
        total += item
    return total
```

---

Sum/Total Pattern w/ File:

---

```
def foo(filename):
    file_object = open(filename)
    lines = file_object.readlines()
    total = 0
    for line in lines:
        total += int(line)
    return total
```

---