# Reference guide: Python concepts from Course 7

Google Cybersecurity Certificate

			•			
		$\sim$ 1	-1		n	C
U	ᆫ	Ct	ш	U	ш	0

**Comments** 

Conditional statements

**Iterative statements** 

<u>User-defined functions</u>

**Built-in functions** 

Importing module and libraries

String methods

List methods

Additional syntax for working with strings and lists

Regular expressions

File operations

**Parsing** 

## Comments

The following syntax is used to create a comment. (A comment is a note programmers make about the intention behind their code.)

#

Starts a line that contains a Python comment

```
# Print approved usernames
```

Contains a comment that indicates the purpose of the code that follows it is to print approved usernames

# """ (documentation strings)

Starts and ends a multi-line string that is often used as a Python comment; multi-line comments are used when you need more than 79 characters in a single comment

```
The estimate_attempts() function takes in a monthly login attempt total and a number of months and returns their product.
```

Contains a multi-line comment that indicates the purpose of the estimate attempts () function

# Conditional statements

The following keywords and operators are used in conditional statements.

if

Starts a conditional statement

```
if device id != "la858zn":
```

Starts a conditional statement that evaluates whether the device\_id variable contains a value that is not equal to "la858zn"

```
if user in approved list:
```

Starts a conditional statement that evaluates if the user variable contains a value that is also found in the approved list variable

#### elif

Precedes a condition that is only evaluated when previous conditions evaluate to False; previous conditions include the condition in the if statement, and when applicable, conditions in other elif statements

```
elif status == 500:
```

When previous conditions evaluate to False, evaluates if the status variable contains a value that is equal to 500

#### else

Precedes a code section that only evaluates when all conditions that precede it within the conditional statement evaluate to False; this includes the condition in the if statement, and when applicable, conditions in elif statements

```
else:
```

When previous conditions evaluate to False, Python evaluates this else statement

#### and

Requires both conditions on either side of the operator to evaluate to True

```
if username == "bmoreno" and login_attempts < 5:
Evaluates to True if the value in the username variable is equal to "bmoreno"
and the value in the login_attempts variable is less than 5</pre>
```

or

Requires only one of the conditions on either side of the operator to evaluate to  ${\tt True}$ 

```
if status == 100 or status == 102:
```

Evaluates to True if the value in the status variable is equal to 100 or the value in the status variable is equal to 102

#### not

Negates a given condition so that it evaluates to False if the condition is True and to True if it is False

```
if not account_status == "removed"
Evaluates to False if the value in the account_status variable is equal to
"removed" and evaluates to True if the value is the account_status variable
is not equal to "removed"
```

## Iterative statements

The following keywords are used in iterative statements.

#### for

Signals the beginning of a for loop; used to iterate through a specified sequence

```
for username in ["bmoreno", "tshah", "elarson"]:
Signals the beginning of a for loop that iterates through the sequence of elements in the list ["bmoreno", "tshah", "elarson"] using the loop variable username
```

```
for i in range(10):
```

Signals the beginning of a for loop that iterates through a sequence of numbers created by range (10) using the loop variable i

#### while

Signals the beginning of a while loop; used to iterate based on a condition

```
while login attempts < 5:
```

Signals the beginning of a while loop that will iterate as long as the condition that the value of login\_attempts is less than 5 evaluates to True

#### break

Used to break out of a loop

#### continue

Used to skip a loop iteration and continue with the next one

# **User-defined functions**

The following keywords are used when creating user-defined functions.

#### def

Placed before a function name to define a function

```
def greet_employee():
Defines the greet_employee() function

def calculate_fails(total_attempts, failed_attempts):
Defines the calculate_fails() function, which includes the two parameters
of total_attempts and failed_attempts
```

#### return

Used to return information from a function; when Python encounters this keyword, it exits the function after returning the information

```
def calculate_fails(total_attempts, failed_attempts):
    fail_percentage = failed_attempts / total_attempts
    return fail_percentage

Returns the value of the fail_percentage variable from the
calculate_fails() function
```

# **Built-in functions**

The following built-in functions are commonly used in Python.

```
print()
```

Outputs a specified object to the screen

```
print("login success")
Outputs the string "login success" to the screen
```

```
print(9 < 7)
```

Outputs the Boolean value of False to the screen after evaluating whether the integer 9 is less than the integer 7

## type()

Returns the data type of its input

```
print(type(51.1))
```

Returns the data type of float for the input of 51.1

```
print(type(True))
```

Returns the data type of Boolean for the input of True

## range()

Generates a sequence of numbers

```
range(0, 5, 1)
```

Generates a sequence with a start point of 0, a stop point of 5, and an increment of 1; because the start point is inclusive but the stop point is exclusive, the generated sequence is 0, 1, 2, 3, and 4

```
range(5)
```

Generates a sequence with a stop point of 5; when the start point is not specified, it is set at the default value of 0, and when the increment is not specified, it is set at the default value of 1; the generated sequence is 0, 1, 2, 3, and 4

## max()

Returns the largest numeric input passed into it

```
print(max(10, 15, 5))
```

Returns 15 and outputs this value to the screen

```
min()
```

Returns the smallest numeric input passed into it

```
print(min(10, 15, 5))
```

Returns 5 and outputs this value to the screen

```
sorted()
```

Sorts the components of a list (or other iterable)

```
print(sorted([10, 15, 5]))
```

Sorts the elements of the list from smallest to largest and outputs the sorted list of [5, 10, 15] to the screen

```
print(sorted(["bmoreno", "tshah", "elarson"]))
```

Sorts the elements in the list in alphabetical order and outputs the sorted list of ["bmoreno", "elarson", "tshah"] to the screen

## str()

Converts the input object to a string

```
str(10)
```

Converts the integer 10 to the string "10"

## len()

Returns the number of elements in an object

```
print(len("security"))
```

Returns and displays 8, the number of characters in the string "security"

# Importing modules and libraries

The following keyword is used to import a module from the Python Standard Library or to import an external library that has already been installed.

```
import
```

Searches for a module or library in a system and adds it to the local Python environment

```
import statistics
Imports the statistics module and all of its functions from the Python
Standard Library

from statistics import mean
Imports the mean() function of the statistics module from the Python
Standard Library

from statistics import mean, median
Imports the mean() and median() functions of the statistics module from the Python Standard Library
```

# String methods

The following methods can be applied to strings in Python.

```
.upper()
```

Returns a copy of the string in all uppercase letters

```
print("Security".upper())
Returns and displays a copy of the string "Security" as "SECURITY"
```

```
.lower()
```

Returns a copy of the string in all lowercase letters

```
print("Security".lower())
Returns and displays a copy of the string "Security" as "security"
```

```
.index()
```

Finds the first occurrence of the input in a string and returns its location

```
print("Security".index("c"))
```

Finds the first occurrence of the character "c" in the string "Security" and returns and displays its index of 2

## List methods

The following methods can be applied to lists in Python.

```
.insert()
```

Adds an element in a specific position inside the list

```
username_list = ["elarson", "fgarcia", "tshah"]
username_list.insert(2,"wjaffrey")
Adds the element "wjaffrey" at index 2 to the username_list; the list
becomes ["elarson", "fgarcia", "wjaffrey", "tshah"]
```

#### .remove()

Removes the first occurrence of a specific element inside a list

```
username_list = ["elarson", "bmoreno", "wjaffrey", "tshah"]
username_list.remove("elarson")
Removes the element "elarson" from the username_list; the list becomes
["fgarcia", "wjaffrey", "tshah"]
```

## .append()

Adds input to the end of a list

```
username_list = ["bmoreno", "wjaffrey", "tshah"]
username_list.append("btang")
Adds the element "btang" to the end of the username_list; the list
becomes ["fgarcia", "wjaffrey", "tshah", "btang"]
```

## .index()

Finds the first occurrence of an element in a list and returns its index

```
username_list = ["bmoreno", "wjaffrey", "tshah", "btang"]
print(username_list.index("tshah"))
```

Finds the first occurrence of the element "tshah" in the username\_list and returns and displays its index of 2

# Additional syntax for working with strings and lists

The following syntax is useful when working with strings and lists.

## + (concatenation)

Combines two strings or lists together

```
device id = "IT"+"nwp12"
```

Combines the string "IT" with the string "nwp12" and assigns the combined string of "ITnwp12" to the variable device id

```
users = ["elarson", "bmoreno"] + ["tshah", "btang"]
Combines the list ["elarson", "bmoreno"] with the list ["tshah",
"btang"] and assigns the combined list of ["elarson", "bmoreno",
"tshah", "btang"] to the variable users
```

# [] (bracket notation)

Uses indices to extract parts of a string or list

```
print("h32rb17"[0])
```

Extracts the character at index 0, which is ("h"), from the string "h32rb17"

```
print("h32rb17"[0:3])
```

Extracts the slice [0:3], which is ("h32"), from the string "h32rb17"; the first index in the slice (0) is included in the slice but the second index in the slice (3) is excluded

```
username_list = ["elarson", "fgarcia", "tshah"]
print(username list[2])
```

Extracts the element at index 2, which is ("tshah"), from the username list

# Regular expressions

The following re module function and regular expression symbols are useful when searching for patterns in strings.

```
re.findall()
```

Returns a list of matches to a regular expression

```
import re
re.findall("a53", "a53-32c .E")
```

Returns a list of matches to the regular expression pattern "a53" in the string "a53-32c"; returns the list ["a53"]

\w

Matches with any alphanumeric character; also matches with the underscore ( )

```
import re
re.findall("\w", "a53-32c .E")
```

Returns a list of matches to the regular expression pattern " $\w$ " in the string "a53-32c"; matches to any alphanumeric character and returns the list ["a", "5", "3", "3", "c", "E"]

.

Matches to all characters, including symbols

```
import re
re.findall(".", "a53-32c .E")
```

Returns a list of matches to the regular expression pattern "." in the string "a53-32c .E"; matches to all characters and returns the list ["a", "5", "3", "-", "3", "2", "c", " ", ".", "E"]

\d

Matches to all single digits

```
import re
re.findall("\d", "a53-32c .E")
```

Returns a list of matches to the regular expression pattern "\d" in the string "a53-32c"; matches to all single digits and returns the list ["5", "3", "3", "2"]

\s

Matches to all single spaces

```
import re
re.findall("\s", "a53-32c .E")
```

Returns a list of matches to the regular expression pattern " $\s$ " in the string "a53-32c". E"; matches to all single spaces and returns the list [" "]

١.

Matches to the period character

```
import re
re.findall("\.", "a53-32c .E")
```

Returns a list of matches to the regular expression pattern "\." in the string "a53-32c"; matches to all instances of the period character and returns the list ["."]

+

Represents one or more occurrences of a specific character

```
import re
re.findall("\w+", "a53-32c .E")
```

Returns a list of matches to the regular expression pattern " $\w+$ " in the string "a53-32c". E"; matches to one or more occurrences of any alphanumeric character and returns the list ["a53", "32c", "E"]

Represents, zero, one or more occurrences of a specific character

```
import re
re.findall("\w*", "a53-32c .E")
Returns a list of matches to the regular expression pattern "\w*" in the string
"a53-32c .E"; matches to zero, one or more occurrences of any
alphanumeric character and returns the list ["a53", " ", "32c", " ", "
", "E"]
```

Represents a specified number of occurrences of a specific character; the number is specified within the curly brackets

```
import re
re.findall("\w{3}", "a53-32c .E")
```

Returns a list of matches to the regular expression pattern " $\w{3}$ " in the string "a53-32c"; matches to exactly three occurrences of any alphanumeric character and returns the list ["a53", "32c"]

# File operations

The following functions, methods, and keywords are used with operations involving files.

#### with

Handles errors and manages external resources

```
with open("logs.txt", "r") as file:
```

Used to handle errors and manage external resources while opening a file; the variable file stores the file information while inside of the with statement; manages resources by closing the file after exiting the with statement

```
open()
```

Opens a file in Python

```
with open("login attempts.txt", "r") as file:
```

```
Opens the file "login_attempts.txt" in order to read it ("r") with open ("update_log.txt", "w") as file:

Opens the file "update_log.txt" into the variable file in order to write over its contents ("w")
```

```
with open(import file, "a") as file:
```

Opens the file assigned to the import\_file variable into the variable file in order to append information to the end of it ("a")

as

Assigns a variable that references another object

```
with open("logs.txt", "r") as file:
```

Assigns the file variable to reference the output of the open () function

```
.read()
```

Converts files into strings; returns the content of an open file as a string by default

```
with open("login_attempts.txt", "r") as file:
    file_text = file.read()
```

Converts the file object referenced in the file variable into a string and then stores this string in the file\_text variable

```
.write()
```

Writes string data to a specified file

```
with open("access_log.txt", "a") as file:
    file.write("jrafael")
```

Writes the string "jrafael" to the "access\_log.txt" file; because the second argument in the call to the open() function is "a", this string is appended to the end of the file

# **Parsing**

The following methods are useful when parsing data.

```
.split()
```

Converts a string into a list; separates the string based on the character that is passed in as an argument; if an argument is not passed in, it will separate the string each time it encounters whitespace characters such as a space or return

```
approved_users = "elarson, bmoreno, tshah".split(",")

Converts the string "elarson, bmoreno, tshah" into the list

["elarson", "bmoreno", "tshah"] by splitting the string into a separate list element at each occurrence of the "," character

removed_users = "wjaffrey jsoto abernard".split()

Converts the string "wjaffrey jsoto abernard" into the list

["wjaffrey", "jsoto", "abernard"] by splitting the string into a separate list element at each space
```

# .join()

Concatenates the elements of an iterable into a string; takes the iterable to be concatenated as an argument; is appended to a character that will separate each element once they are joined into a string

```
approved_users = ",".join(["elarson", "bmoreno", "tshah"])
Concatenates the elements of the list ["elarson", "bmoreno", "tshah"]
into the string "elarson, bmoreno, tshah", separating each element with
the "," character within the string
```