SKILLS NETWORK

Python Cheat Sheet: The Basics

Python Data Types

String
Series of characters or data stored as text

```
my_string = "Hello"
```

String Operations

```
# returns the string with all uppercase letters
my_string.upper()
```

```
# returns the length of a string
len(my_string)
```

```
# returns the index of the first instance of the string inside the
# subject string, otherwise -1
my_string.find('l')
```

```
# replaces any instance of the first string with the second in my_string
my_string.replace('H', 'C')
```

Integer
A whole number

```
my_integer = 12321
```

Float
A decimal number

```
my_decimal = 3.14
```

Boolean
Discrete value true or false

```
a = True
b = False
```

Dictionary
Changeable collection of key-value pairs

```
my_dictionary = {'banana': 1, 12: 'laptop', (0,0):'center'}
```

Dictionary Operations

```
# Access value using key
my_dictionary['banana']
```

```
# Get all keys in a dictionary as a list
my_dictionary.keys()
```

```
# Get all values in a dictionary as a list
my_dictionary.values()
```

Tuple
Unchangeable collection of objects

```
tup = (1, 3.12, False, "Hi")
```

List
Changeable collection of objects

```
my_collection = [1, 1, 3.12, False, "Hi"]
```

List Operations

```
# returns the length of a list
len(my_collection)
```

```
# Add multiple items to a list
my_collection.extend(["More", "Items"])
```

```
# Add a single item to a list
my_collection.append("Single")
```

```
# Delete the object of a list at a specified index
del(my_collection[2])
```

```
# Clone a list
clone = my_collection[:]
```

```
# Concatenate two lists
my_collection_2 = ["a", "b", "c"]
my_collection_3 = my_collection + my_collection_2
```

```
# Calculate the sum of a list of ints or floats
number_collection = [1,2,3,4.5]
sum(number_collection)
```

```
# Check if an item is in a list, returns Boolean
item in my_collection
# Check if an item is not in a list, returns Boolean
item not in my_collection
```

Set
Unordered collection of unique objects

```
a = {100, 3.12, False, "Bye"}
b = {100, 3.12, "Welcome"}
```

Set Operations

```
# Convert a list to a set
my_set = set([1,1,2,3])
```

```
# Add an item to the set
a.add(4)
```

```
# Remove an item from a set
a.remove("Bye")
```

```
# Returns set a minus b
a.difference(b)
```

```
# Returns intersection of set a and b
a.intersection(b)
```

```
# Returns the union of set a and b
a.union(b)
```

```
# Returns True if a is a subset of b, false otherwise
a.issubset(b)
```

```
# Returns True if a is a superset of b, false otherwise
a.issuperset(b)
```

The method `Split` splits the string at the specified separator, and returns a list:

```
#Split the substring into list
name = "Michael Jackson"
split_string = (name.split())
split_string
```

```
['Michael', 'Jackson']
```

```
'Gusty,Baby,junggo'.split(',')
```

```
['Gusty', 'Baby', 'junggo']
```

Indexing
Accessing data from a string, list, or tuple using an element number

```
my_string[element_number]
my_collection[element_number]
my_tup[element_number]
```

Slicing
Accessing a subset of data from a string, list, or tuple using element numbers from start to stop -1

```
my_string[start:stop]
my_collection[start:stop]
my_tup[start:stop]
```

Comparison Operators

Comparison Operators compare operands and return a result of true or false

Equal

```
a == b
```

Less Than

```
a < b
```

Greater Than

```
a > b
```

Greater Than or Equal

```
a >= b
```

Less Than or Equal

```
a <= b
```

Not Equal

```
a != b
```

Python Operators

- +: Addition
- -: Subtraction
- *: Multiplication
- /: division
- //: Integer Division (Result rounded to the nearest integer)

Conditional Operators

Conditional Operators evaluate the operands and produce a true of false result

And - returns true if both statement a and b are true, otherwise false

```
a and b
```

Or - returns true if either statement a or b are true, otherwise false

```
a or b
```

Not - returns the opposite of the statement

```
not a
```

SKILLS NETWORK

Python Cheat Sheet: The

Mode	Syntax	Description
'r'	"r"	Read mode. Opens an existing file for reading. Raises an error if the file doesn't exist.
'w'	"w"	Write mode. Creates a new file for writing. Overwrites the file if it already exists.
'a'	"a"	Append mode. Opens a file for appending data. Creates the file if it doesn't exist.
'x'	"x"	Exclusive creation mode. Creates a new file for writing but raises an error if the file already exists.
'rb'	"rb"	Read binary mode. Opens an existing binary file for reading.
'wb'	"wb"	Write binary mode. Creates a new binary file for writing.
'ab'	"ab"	Append binary mode. Opens a binary file for appending data.
'xb'	"xb"	Exclusive binary creation mode. Creates a new binary file for writing but raises an error if it already exists.

'rt'	"rt"	Read text mode. Opens an existing text file for reading. (Default for text files)
'wt'	"wt"	Write text mode. Creates a new text file for writing. (Default for text files)
'at'	"at"	Append text mode. Opens a text file for appending data. (Default for text files)
'xt'	"xt"	Exclusive text creation mode. Creates a new text file for writing but raises an error if it already exists.
'r+'	"r+"	Read and write mode. Opens an existing file for both reading and writing.
'w+'	"w+"	Write and read mode. Creates a new file for reading and writing. Overwrites the file if it already exists.
'a+'	"a+"	Append and read mode. Opens a file for both appending and reading. Creates the file if it doesn't exist.
'x+'	"x+"	Exclusive creation and read/write mode. Creates a new file for reading and writing but raises an error if it already exists.

Loops

For Loops

```
for x in range(x):
    # Executes loop x number of times
```

```
for x in iterable:
    # Executes loop for each object in an iterable like a string, tuple,
list, or set
```

While Loops

```
while statement:
    # Executes the loop while statement is true
```

Conditional Statements

```
if statement_1:
  # Execute of statement_1 is true
elif statement_2:
  # Execute if statement_1 is false and statement_2 is true
else:
  # Execute if all previous statements are false
```

Try/Except

```
try:
   # Code to try to execute
except a:
   # Code to execute if there is an error of type a
except b:
   # Code to execute if there is an error of type b
except:
   # Code to execute if there is any exception that has not been handled
else:
   # Code to execute if there is no exception
```

Error Types

- IndexError - When an index is out of range
- NameError - When a variable name is not found
- SyntaxError - When there is an error with how the code is written
- ZeroDivisionError - When your code tries to divide by zero

Range

Produce an iterable sequence from 0 to stop-1

```
range(stop)
```

Produce an interable sequence from start to stop-1 incrementing by step

```
range(start, stop, step)
```

Webscraping

```
# Import BeautifulSoup
from bs4 import BeautifulSoup
```

```
# Parse HTML stored as a string
soup = BeautifulSoup(html, 'html5lib')
```

```
# Returns formatted html
soup.prettify()
```

```
# Find the first instance of an HTML tag
soup.find(tag)
```

```
# Find all instances of an HTML tag
soup.find_all(tag)
```

Requests

```
# Import the requests library
import requests
```

```
# Send a get requests to the url with optional parameters
response = requests.get(url, parameters)
```

```
# Get the url of the response
response.url
# Get the status code of the response
response.status_code
# Get the headers of the request
response.request.headers
# Get the body of the requests
response.request.body
# Get the headers of the response
response.headers
# Get the content of the response in text
response.text
# Get the content of the response in json
response.json()
```

```
# Send a post requests to the url with optional parameters
requests.post(url, parameters)
```

Functions

```
# Create a function
def function_name(optional_parameter_1, optional_prameter_2):
     # code to execute
     return optional_output
```

```
# Calling a function
output = function_name(parameter_1, parameter_2)
```

Working with Files

Reading a File

```
# Opens a file in read mode
file = open(file_name, "r")
# Returns the file name
file.name
# Returns the mode the file was opened in
file.mode
file.seek(10)
# Reads the contents of a file
file.read()
```

```
# Reads a certain number of characters of a file
file.read(characters)
characters = file.read(5)  # Read the next 5 characters
```

```
# Read a single line of a file
file.readline()
```

```
# Read all the lines of a file and stores it in a list
file.readlines()
```

```
# Closes a file
file.close()
```

Writing to a File

```
# Opens a file in write mode
file = open(file_name, "w")
```

```
# Writes content to a file
file.write(content)
```

```
# Adds content to the end of a file
file.append(content)
```

- `.tell()` - returns the current position in bytes
- `.seek(offset,from)` - changes the position by 'offset' bytes with respect to 'from'. From can take the value of 0,1,2 corresponding to beginning, relative to current position and end

Objects and Classes

```
# Creating a class
class class_name:
     def __init__(self. optional_parameter_1, optional_parameter_2):
          self.attribute_1 = optional_parameter_1
          self.attribute_2 = optional_parameter_2

     def method_name(self, optional_parameter_1):
          # Code to execute
          return optional_output
```

```
# Create an instance of a class
object = class_name(parameter_1, parameter_2)
```

```
# Calling an object method
object.method_name(parameter_3)
```

```
# New line escape sequence

print(" Michael Jackson \n is the best" )
```

Similarly, back slash "t" represents a tab:

```
# Tab escape sequence

print(" Michael Jackson \t is the best" )
```

If you want to place a back slash in your string, use a double back slash:

```
# Include back slash in string

print(" Michael Jackson \\ is the best" )
```

We can also place an "r" before the string to display the backslash:

```
# r will tell python that string will be display as raw string

print(r" Michael Jackson \ is the best" )
```

library

RegEx

```
import re
```

Special Sequence	Meaning	Example
\d	Matches any digit character (0-9)	"123" matches "\d\d\d"
\D	Matches any non-digit character	"hello" matches "\D\D\D\D\D"
\w	Matches any word character (a-z, A-Z, 0-9, and _)	"hello_world" matches "\w\w\w\w\w\w\w\w\w\w\w"
\W	Matches any non-word character	"@#$%" matches "\W\W\W\W"
\s	Matches any whitespace character (space, tab, newline, etc.)	"hello world" matches "\w\w\w\w\w\s\w\w\w\w\w"
\S	Matches any non-whitespace character	"hello_world" matches "\S\S\S\S\S\S\S\S\S\S"
\b	Matches the boundary between a word character and a non-word character	"cat" matches "\bcat\b" in "The cat sat on the mat"
\B	Matches any position that is not a word boundary	"cat" matches "\Bcat\B" in "category" but not in "The cat sat on the mat"

```
str1= "The quick brown fox jumps over the lazy dog."

# Write your code below and press Shift+Enter to execute
re.sub(r"bear",'fox',str1)
```

```
'The quick brown fox jumps over the lazy dog.'
```

```
str2= "How much wood would a woodchuck chuck, if a woodchuck could chuck wood?"

# Write your code below and press Shift+Enter to execute
re.findall('woo',str2)
```

```
['woo', 'woo', 'woo', 'woo']
```

```
s3 = "House number- 1105"
# Write your code below and press Shift+Enter to execute
re.search(r'\d\d\d\d',s3)
```

```
<re.Match object; span=(14, 18), match='1105'>
```

Care about compare letters

Char.	ASCII	Char.	ASCII	Char.	ASCII	Char.	ASCII
A	65	N	78	a	97	n	110
B	66	O	79	b	98	o	111
C	67	P	80	c	99	p	112
D	68	Q	81	d	100	q	113
E	69	R	82	e	101	r	114
F	70	S	83	f	102	s	115
G	71	T	84	g	103	t	116
H	72	U	85	h	104	u	117
I	73	V	86	i	105	v	118
J	74	W	87	j	106	w	119
K	75	X	88	k	107	x	120
L	76	Y	89	l	108	y	121
M	77	Z	90	m	109	z	122

```
# Compare characters

'BA' > 'AB'
```

```
True
```

Note: Upper Case Letters have different ASCII code than Lower Case Letters, which means the comparison between the letters in Python is case-sensitive.

Type of Except

Table 1.1 Built-in exceptions in Python

S. No	Name of the Built-in Exception	Explanation
1.	SyntaxError	It is raised when there is an error in the syntax of the Python code.
2.	ValueError	It is raised when a built-in method or operation receives an argument that has the right data type but mismatched or inappropriate values.
3.	IOError	It is raised when the file specified in a program statement cannot be opened.
4	KeyboardInterrupt	It is raised when the user accidentally hits the Delete or Esc key while executing a program due to which the normal flow of the program is interrupted.
5	ImportError	It is raised when the requested module definition is not found.
6	EOFError	It is raised when the end of file condition is reached without reading any data by input().
7	ZeroDivisionError	It is raised when the denominator in a division operation is zero.
8	IndexError	It is raised when the index or subscript in a sequence is out of range.
9	NameError	It is raised when a local or global variable name is not defined.
10	IndentationError	It is raised due to incorrect indentation in the program code.
11	TypeError	It is raised when an operator is supplied with a value of incorrect data type.
12	OverFlowError	It is raised when the result of a calculation exceeds the maximum limit for numeric data type.

Input

↓ Type

```
numerator = int(input("Enter Value of numerator"))
denominator = int(input("Enter Value of denominator"))
```

Class & Object

```
#Type your code here
class vel:
    color = 'white'
    def __init__(self, max,mile):
        self.max = max
        self.mile = mile
    def seat(self, seat_capa):
        self.seat_capa = seat_capa
    def display(self):
        print(self.color,' ',self.max,' ',self.mile,' ',self.seat_capa)
```

To use class

```
a = vel(200,20)
a.seat(5)
a.display()
```

```
white   200   20   5
```

sort()	The `sort()` method is used to sort the elements of a list in ascending order. If you want to sort the list in descending order, you can pass the `reverse=True` argument to the `sort()` method.	Example 1:
		`my_list = [5, 2, 8, 1, 9]`
		`my_list.sort()`
		`print(my_list)`
		`# Output: [1, 2, 5, 8, 9]`
		Example 2:
		`my_list = [5, 2, 8, 1, 9]`
		`my_list.sort(reverse=True)`
		`print(my_list)`
		`# Output: [9, 8, 5, 2, 1]`