**Python Cheat Sheet**

**Source: The Python Mega Course by Ardit Sulce**

Course link: <http://udemy.com/the-python-mega-course>

**Python Data Types**

**Integers** are used to represent whole numbers:



rank = 10

eggs = 12

people = 3

**Floats** represent decimal numbers:



temperature = 10.2

rainfall = 5.98

elevation = 1031.88

**Strings** represent text:



message = "Welcome to our online shop!"

name = "John"

serial = "R001991981SW"

**Lists** represent arrays of values that may change during the course of theprogram:



members = ["Sim Soony", "Marry Roundknee", "Jack Corridor"] pixel\_values = [252, 251, 251, 253, 250, 248, 247]

**Dictionaries** represent pairs of keys and values:



phone\_numbers = {"John Smith": "+37682929928", "Marry Simpons": "+423998200919"}

volcano\_elevations = {"Glacier Peak": 3213.9, "Rainer": 4392.1}

**Keys** of a dictionary can be extracted with:



phone\_numbers.keys()

**Values** of a dictionary can be extracted with:



phone\_numbers.values()

**Tuples** represent arrays of values that are not to be changed during the courseof the program:



vowels = ('a', 'e', 'i', 'o', 'u')

one\_digits = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

You can get a list of **attributes** of a data type has using:



dir(str)

dir(list)

dir(dict)

You can get a list of Python **builtin functions** using:



dir(\_\_builtins\_\_)

You can get the **documentation** of a Python data type using:



help(str)

help(str.replace)

help(dict.values)

**Operations with Data Types**

Lists, strings, and tuples have a **positive index** system:



["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]

0 1 2 3 4 5 6

And they have a **negative index** system as well:



["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]

-7 -6 -5 -4 -3 -2 -1

In a list, the **2nd**, **3rd**, and **4th** items can be accessed with:



days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"] days[1:4]

Output: ['Tue', 'Wed', 'Thu']

**First three items of a list**:



days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"] days[:3]

Output:['Mon', 'Tue', 'Wed']

**Last three items of a list**:



days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"] days[-3:]

Output: ['Fri', 'Sat', 'Sun']

**Everything but the last**:



days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"] days[:-1]

Output: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']

**Everything but the last two**:



days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"] days[:-2]

Output: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri']

A dictionary **value** can be accessed using its corresponding dictionary **key**:



phone\_numbers = {"John":"+37682929928","Marry":"+423998200919"}

phone\_numbers["Marry"]

Output: '+423998200919'

**Functions and Conditionals**

Define **functions**:



def cube\_volume(a):

return a \* a \* a

Write **if-else conditionals**:



message = "hello there"

if "hello" in message:

print("hi")

else:

print("I don't understand")

Write **if-elif-else conditionals:**

message = "hello there"

if "hello" in message:

print("hi")

elif "hi" in message:

print("hi")

elif "hey" in message:

print("hi")

else:

print("I don't understand")

Use the and operator to check if **both conditions** are True at the same time:

x = 1

y = 1

if x == 1 and y==1:

print("Yes")

else:

print("No")

Use the or operator to check if **at least one condition** is True:

x = 1

y = 2

if x == 1 or y==2:

print("Yes")

else:

print("No")

Check if a value is of a particular **type** with **isinstance**:



isinstance("abc", str)

isinstance([1, 2, 3], list)

or directly:

type("abc") == str

type([1, 2, 3]) == lst

**Processing User Input**

A Python program can get **user input** via the input function:



The **input function** halts the execution of the program and gets text input from the user**:**



name = input("Enter your name: ")

The input function converts any **input to a string**, but you can convert it back to int or float:



experience\_months = input("Enter your experience in months: ")

experience\_years = int(experience\_months) / 12

You can also **format strings** with:



name = "Sim"

experience\_years = 1.5

print("Hi {}, you have {} years of experience".format(name, experience\_years))

Output: Hi Sim, you have 1.5 years of experience.

**Python Loops**

A **for-loop** is useful to repeatedly execute a block of code.



You can create a **for-loop** like so:



for letter in 'abc':

print(letter.upper())

Output:

A B C

As you can see, the for-loop repeatedly converted all the items of 'abc' to uppercase.

The name after for (e.g. letter ) is just a variable name



You can loop over **dictionary keys** as follows:



phone\_numbers = {"John Smith":"+37682929928","Marry Simpons":"+423998200919"} for value in phone\_numbers.keys():

print(value)

Output:

John Smith Marry Simpsons

You can loop over **dictionary values**:



phone\_numbers = {"John Smith":"+37682929928","Marry Simpons":"+423998200919"} for value in phone\_numbers.values():

print(value)

Output:

+37682929928 +423998200919

You can loop over **dictionary items**:



phone\_numbers = {"John Smith":"+37682929928","Marry Simpons":"+423998200919"} for key, value in phone\_numbers.items():

print(key, value)

Output:

John Smith +37682929928

Marry Simpons +423998200919

We also have **while-loops**. The code under a while-loop will run as long as the while-loop condition is true:



while datetime.datetime.now() < datetime.datetime(2090, 8, 20, 19, 30, 20):

print("It's not yet 19:30:20 of 2090.8.20")

The loop above will print out the string inside print() over and over again until the 20th of August, 2090.



**List Comprehensions**

A list comprehension is an expression that creates a list by iterating over another container.



A **basic** list comprehension:



[i\*2 for i in [1, 5, 10]]

Output: [2, 10, 20]

List comprehension with **if** condition:



[i\*2 for i in [1, -2, 10] if i>0]

Output: [2, 20]

List comprehension with an **if and else** condition:



[i\*2 if i>0 else 0 for i in [1, -2, 10]]

Output: [2, 0, 20]

**More on Functions**

Functions can have more than one **parameter**:



def volume(a, b, c):

return a \* b \* c

Functions can have **default** parameters (e.g. coefficient ):

def converter(feet, coefficient = 3.2808):

meters = feet / coefficient

return meters

print(converter(10))

Output: 3.0480370641306997

Arguments can be passed as **non-keyword** (positional) arguments (e.g. a )

or **keyword** arguments (e.g. b=2 and c=10 ):

def volume(a, b, c):

return a \* b \* c

print(volume(1, b=2, c=10))

An **args** parameter allows the function to be called with an arbitrary number of non-keyword arguments:



def find\_max(\*args):

return max(args)

print(find\_max(3, 99, 1001, 2, 8))

Output: 1001

A **\*kwargs** parameter allows the function to be called with an arbitrary number of keyword arguments:



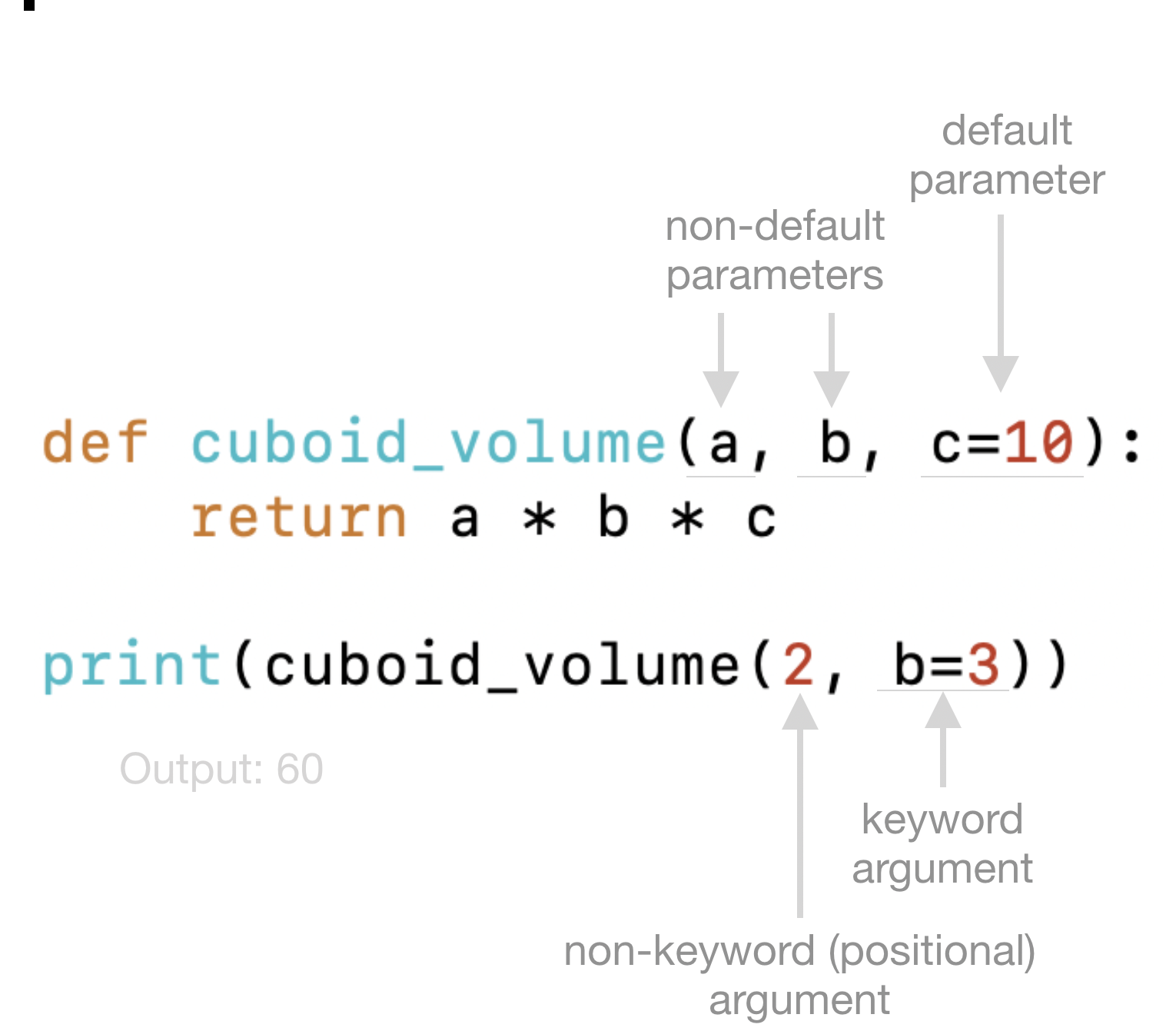
def find\_winner(\*\*kwargs):

return max(kwargs, key = kwargs.get)

print(find\_winner(Andy = 17, Marry = 19, Sim = 45, Kae = 34))

Output: Sim

Here is a visual summary of function elements:



**File Processing in Python**

You can **read** an existing file with Python:



with open("file.txt") as file:

content = file.read()

You can **create** a new file with Python and **write** some text on it:



with open("file.txt", "w") as file:

content = file.write("Sample text")

You can **append** text to an existing file without overwriting it:



with open("file.txt", "a") as file:

content = file.write("More sample text")

You can both **append and read** a file with:



with open("file.txt", "a+") as file:

content = file.write("Even more sample text")

file.seek(0)

content = file.read()

**Python Modules**

**Builtin objects** are all objects that are written inside the Pythoninterpreter in C language.



**Builtin modules** contain builtins objects.



Some builtin objects are not immediately available in the global namespace. They are parts of a builtin module. To use those objects the module needs to be **imported** first. E.g.:



import time

time.sleep(5)

**A list of all builtin modules** can be printed out with:



import sys

sys.builtin\_module\_names

**Standard libraries** is a jargon that includes both builtin modules writtenin C and also modules written in Python.



**Standard libraries** written in Python reside in the Python installation



directory as *.py* files. You can find their directory path with” sys.” prefix

**Packages** are a collection of*.py*modules.



**Third-party libraries** are packages or modules written by third-partypersons (not the Python core development team).



Third-party libraries can be **installed** from the terminal/command line:



Windows:

pip install pandas or use python -m pip install pandas if that doesn't

work.

Mac and Linux:



pip3 install pandas or use python3 -m pip install pandas if that doesn't

work.