Python Cheat Sheet

Operators

* Arithmetic operators

+ Addition x + y

- Subtraction x - y

\* Multiplication x \* y

/ Division x / y

% Modulus x % y

\*\* Exponentiation x \*\* y

// Floor division x // y

* Comparison Operators

== Equal x == y

!= Not equal x != y

> Greater than x > y

< Less than x < y

>= Greater than or equal to x >= y

<= Less than or equal to x <= y

* Assignment Operators

= x = 5 x = 5

+= x += 3 x = x + 3

-= x -= 3 x = x - 3

\*= x \*= 3 x = x \* 3

/= x /= 3 x = x / 3

%= x %= 3 x = x % 3

//= x //= 3 x = x // 3

\*\*= x \*\*= 3 x = x \*\* 3

&= x &= 3 x = x & 3

|= x |= 3 x = x | 3

^= x ^= 3 x = x ^ 3

>>= x >>= 3 x = x >> 3

<<= x <<= 3 x = x << 3

* Logical Operators

and Returns True if both statements are true x < 5 and x < 10

or Returns True if one of the statements is true x < 5 or x < 4

not Reverse the result, returns False if the result is true not(x < 5 and x < 10)

* Identity Operators

is Returns True if both variables are the same object x is y

is not Returns True if both variables are not the same object x is not y

* Membership Operators

in Returns True if a sequence with the specified value

is present in the object x in y

not in Returns True if a sequence with the specified value

is not present in the object x not in y

* Bitwise Operators

& AND Sets each bit to 1 if both bits are 1

| OR Sets each bit to 1 if one of two bits is 1

^ XOR Sets each bit to 1 if only one of two bits is 1

~ NOT Inverts all the bits

<< Zero fill left shift Shift left by pushing zeros in from the right

and let the leftmost bits fall off

>> Signed right shift Shift right by pushing copies of the leftmost

bit in from the left, and let the rightmost

bits fall off

Data Types

* Integers

558787

-6868

6655

* Float

575.9

68686.0

-57575.889

* Strings

‘hello’

“Hello”

‘he said “my number is 10.7”’

* Booleans

true (or 1)

false (or 0)

Print Command

* Print in console

print()

print(‘hello’)

print(688.5)

print(‘hello world’, 7.3)

- Print with no carriage return (as python will always try to create new lines with each print line)

print(‘hello’, 66.8, end=’\n’)

print(‘hello’)

=

Hello 66.8 hello

F strings

- Embed code into strings

name = ‘James’

print(f’hello {name} my friend’)

print(f’your score is {4 + 8}’)

=

hello james my friend

your score is 12

Variable

* Basic var (should only use snake\_case to name e.g. var\_name)

hello = 7

hello\_apple = ‘hi’

* Accessing var

Hello

User prompts

* Input (result will be a string)

input(‘Age: ’)

user\_age=input(‘Age: ’)

print(user\_age)

=

Age: 66

66

* Integer input

user\_age=input(‘Age: ’)

print(int(user\_age))

=

Age: 66

66

* Float input

user\_age=input(‘Age: ’)

print(float(user\_age))

=

Age: 66

66.0

user\_age=input(‘Age: ’)

print(float(user\_age))

=

Age: 67.6

67.6

Data type finding

* Type

var\_name = ‘hello world’

print(type(var\_name))

=

<class ‘str’>

var\_name = 77.9

print(type(var\_name))

=

<class ‘float’>

Method

* Capitalise whole string

var\_name\_one = ‘hello world’

print(var\_name\_one.upper())

var\_name\_two = ‘hello world’.upper()

print(var\_name\_two)

* Lowerletter the whole string

.lower()

* Count

var\_name\_one = ‘hello world’

print(var\_name\_one.count(‘l’))

print(var\_name\_one.count(‘L’))

=

3

0

* Other methods for strings

capitalize() Converts the first character to upper case

casefold() Converts string into lower case

center() Returns a centered string

count() Returns the number of times a specified value occurs in a string

encode() Returns an encoded version of the string

endswith() Returns true if the string ends with the specified value

expandtabs() Sets the tab size of the string

find() Searches the string for a specified value and returns the position of where it was found

format() Formats specified values in a string

format\_map() Formats specified values in a string

index() Searches the string for a specified value and returns the position of where it was found

isalnum() Returns True if all characters in the string are alphanumeric

isalpha() Returns True if all characters in the string are in the alphabet

isascii() Returns True if all characters in the string are ascii characters

isdecimal() Returns True if all characters in the string are decimals

isdigit() Returns True if all characters in the string are digits

isidentifier() Returns True if the string is an identifier

islower() Returns True if all characters in the string are lower case

isnumeric() Returns True if all characters in the string are numeric

isprintable() Returns True if all characters in the string are printable

isspace() Returns True if all characters in the string are whitespaces

istitle() Returns True if the string follows the rules of a title

isupper() Returns True if all characters in the string are upper case

join() Converts the elements of an iterable into a string

ljust() Returns a left justified version of the string

lower() Converts a string into lower case

lstrip() Returns a left trim version of the string

maketrans() Returns a translation table to be used in translations

partition() Returns a tuple where the string is parted into three parts

replace() Returns a string where a specified value is replaced with a specified value

rfind() Searches the string for a specified value and returns the last position of where it was found

rindex() Searches the string for a specified value and returns the last position of where it was found

rjust() Returns a right justified version of the string

rpartition() Returns a tuple where the string is parted into three parts

rsplit() Splits the string at the specified separator, and returns a list

rstrip() Returns a right trim version of the string

split() Splits the string at the specified separator, and returns a list

splitlines() Splits the string at line breaks and returns a list

startswith() Returns true if the string starts with the specified value

strip() Returns a trimmed version of the string

swapcase() Swaps cases, lower case becomes upper case and vice versa

title() Converts the first character of each word to upper case

translate() Returns a translated string

upper() Converts a string into upper case

zfill() Fills the string with a specified number of 0 values at the beginning

* Other methods for lists

append() Adds an element at the end of the list

clear() Removes all the elements from the list

copy() Returns a copy of the list

count() Returns the number of elements with the specified value

extend() Add the elements of a list (or any iterable), to the end of the current list

index() Returns the index of the first element with the specified value

insert() Adds an element at the specified position

pop() Removes the element at the specified position

remove() Removes the first item with the specified value

reverse() Reverses the order of the list

sort() Sorts the list

* Other methods for tuples

count() Returns the number of times a specified value occurs in a tuple

index() Searches the tuple for a specified value and returns the position of where it was found

* Other methods for dictionaries

clear() Removes all the elements from the dictionary

copy() Returns a copy of the dictionary

fromkeys() Returns a dictionary with the specified keys and value

get() Returns the value of the specified key

items() Returns a list containing a tuple for each key value pair

keys() Returns a list containing the dictionary's keys

pop() Removes the element with the specified key

popitem() Removes the last inserted key-value pair

setdefault() Returns the value of the specified key. If the key does not exist: insert the key, with the specified value

update() Updates the dictionary with the specified key-value pairs

values() Returns a list of all the values in the dictionary

* Other methods for sets

add() Adds an element to the set

clear() Removes all the elements from the set

copy() Returns a copy of the set

difference() Returns a set containing the difference between two or more sets

difference\_update() Removes the items in this set that are also included in another, specified set

discard() Remove the specified item

intersection() Returns a set, that is the intersection of two or more sets

intersection\_update() Removes the items in this set that are not present in other, specified set(s)

isdisjoint() Returns whether two sets have a intersection or not

issubset() Returns whether another set contains this set or not

issuperset() Returns whether this set contains another set or not

pop() Removes an element from the set

remove() Removes the specified element

symmetric\_difference() Returns a set with the symmetric differences of two sets

symmetric\_difference\_update() inserts the symmetric differences from this set and another

union() Return a set containing the union of sets

update() Update the set with another set, or any other iterable

* Other methods for working with files

close() Closes the file

detach() Returns the separated raw stream from the buffer

fileno() Returns a number that represents the stream, from the operating system's perspective

flush() Flushes the internal buffer

isatty() Returns whether the file stream is interactive or not

read() Returns the file content

readable() Returns whether the file stream can be read or not

readline() Returns one line from the file

readlines() Returns a list of lines from the file

seek() Change the file position

seekable() Returns whether the file allows us to change the file position

tell() Returns the current file position

truncate() Resizes the file to a specified size

writable() Returns whether the file can be written to or not

write() Writes the specified string to the file

writelines() Writes a list of strings to the file

Chained conditionals

* or

x = true or false

print(x)

=

true

x = true or true

print(x)

=

true

x = false or false

print(x)

=

false

* and

x = false and false

print(x)

=

false

x = false and true

print(x)

=

false

x = true and false

print(x)

=

false

x = true and true

print(x)

=

false

* not

x = not true

print(x)

=

false

x = not false

print(x)

=

true

if / else / elif statements

* if

if x == 24:

print(“it’s 24!!“)

* else

if x == 24:

print(“it’s 24!!“)

else:

print(“it’s not 24!!“)

* elif

if x == 24:

print(“it’s 24!!“)

elif x == 28:

print(“it’s 28!!“)

elif x == 30:

print(“it’s 30!!“)

else:

print(“it’s not 24, 28 or 30!!“)

Lists (arrays) & tuples

* lists (can be changed with methods like .append())

x = [30,21,6]

* tuples (can’t be changed with methods or edited)

x = (30,21,6)

* Other methods for lists

append() Adds an element at the end of the list

clear() Removes all the elements from the list

copy() Returns a copy of the list

count() Returns the number of elements with the specified value

extend() Add the elements of a list (or any iterable), to the end of the current list

index() Returns the index of the first element with the specified value

insert() Adds an element at the specified position

pop() Removes the element at the specified position

remove() Removes the first item with the specified value

reverse() Reverses the order of the list

sort() Sorts the list

* Other methods for tuples

count() Returns the number of times a specified value occurs in a tuple

index() Searches the tuple for a specified value and returns the position of where it was found

Loops

* For (loops a set amount of times based on a count)

- Range e.g. for x in range(start number [default 0 if not set], stop number, step number - the amount you go loop up or down in [default 1 if not set]):

for x in range(5):

print(x)

=

0

1

2

3

4

for x in range(5,10):

print(x)

=

5

6

7

8

9

for x in range(8, 0, -2):

print(x)

=

8

6

4

2

- lists in for loops

for x in [1,4,-55,6]:

print(x)

=

1

4

-55

6

- lists in for loops

x = [1,4,-55,6]

for i, num in range(len(x)):

print(i, num)

=

0 1

1 4

2 -55

3 6

x = [1,4,-55,6]

for i, num in enumerate(x):

print(i, num)

=

0 1

1 4

2 -55

3 6

* While (loops while a set value equals true and stops when it’s false)

while x < 5:

print(‘Running’)

x += 1

break

=

Running

Running

Running

Running

Running

while true:

print(‘Running’)

x += 1

if x = 5:3

break

=

Running

Running

Running

Running

Running

Slice Operator

* Sliced (slices strings, tuples and lists, it kind of works like the for loops: var\_name\_of\_list\_est[start index number(default 0) : stop index number(default end of list/string) : step number it increments in(default 1)]

x = [3,4,6,7,8,3,5,7]

y = [‘hi’, ‘hello’, ‘james’, ‘yes’, ’no’, ‘yeahh’]

s = “hello”

print(x[1:6:2])

print(y[:4])

print(s[::-1])

=

4,7,3

‘hi’, ‘hello’, ‘james’, ‘yes’

olleh

Sets

* Set literal (can’t be left empty or it’ll create a dictionary e.g. ‘{}’, if you need to leave it empty you should use ‘set()’)

x = {7,8,5,7,2}

y = {5,5,7,8,2,4,5,1}

print(x)

print(y)

=

{7,8,5,2}

{5,7,8,2,4,1}

* Add element to set

x = {7,8,5,7,2}

x.add(9)

print(x)

=

{7,8,5,2,9}

* Remove elements from a set

x = {7,8,5,7,2}

y = {5,5,7,8,2,4,5,1}

x.remove(8)

y.remove(5)

print(x)

print(y)

=

{7,5,2}

{7,8,2,4,1}

* Look up number in sets(and lists but the sets are much faster than looking up in lists)

x = {7,8,5,7,2}

y = [5,5,7,8,2,4,5,1]

print(9 in x)

print(7 in x)

print(8 in y)

=

False

True

True

* Adding two sets together

x = {7,8,5,7,2}

y = {5,5,7,8,2,4,5,1}

print(x.union(y))

=

{7,8,5,2,4,1}

* Find the difference of two sets(you’re actually finding why the one set is different to the other)

x2 = {7,8,5,7,2,9}

x = {7,8,5,7,2}

y = {5,5,7,8,2,4,5,1}

print(x2.difference(y))

print(x.difference(y))

print(y.difference(x))

=

{9}

{}

{4,1}

* Find the elements that are the same in two sets

x = {7,8,5,7,2}

y = {5,5,7,8,2,4,5,1}

print(x.intersection(y))

print(y.intersection(x))

=

{7,8,5,2}

{7,8,5,2}

* Other methods for sets

add() Adds an element to the set

clear() Removes all the elements from the set

copy() Returns a copy of the set

difference() Returns a set containing the difference between two or more sets

difference\_update() Removes the items in this set that are also included in another, specified set

discard() Remove the specified item

intersection() Returns a set, that is the intersection of two or more sets

intersection\_update() Removes the items in this set that are not present in other, specified set(s)

isdisjoint() Returns whether two sets have a intersection or not

issubset() Returns whether another set contains this set or not

issuperset() Returns whether this set contains another set or not

pop() Removes an element from the set

remove() Removes the specified element

symmetric\_difference() Returns a set with the symmetric differences of two sets

symmetric\_difference\_update() inserts the symmetric differences from this set and another

union() Return a set containing the union of sets

update() Update the set with another set, or any other iterable

Dictionaries

* Calling on key value pairs(var\_name = {key: data type})

x = {‘key1’: 8}

print(x[‘key1’])

print(x)

=

8

{‘key1’: 8}

* Adding a new key value pair

x = {‘key1’: 8}

x[‘key2’] = ‘hello’

print(x)

=

{‘key1’: 8, ‘key2’: ‘hello’}

* Check if a key name is in the dictionary

x = {‘key1’: 8, ‘key2’: ‘hello’, ‘key3’: [5,6,8]}

print(‘key2’ in x)

print(‘key5’ in x)

=

True

False

* Get all values from dictionary(and put them in a list)

x = {‘key1’: 8, ‘key2’: ‘hello’, ‘key3’: [5,6,8]}

print(x.values())

print(list(x.values()))

=

8,‘hello’,[5,6,8]

[8,‘hello’,[5,6,8]]

* Get all keys from dictionary(and put them in a list)

x = {‘key1’: 8, ‘key2’: ‘hello’, ‘key3’: [5,6,8]}

print(x.keys())

print(list(x.keys()))

=

‘key1’,‘key2’,‘key3’

[‘key1’,‘key2’,‘key3’]

* Delete a key value pair from dictionary

x = {‘key1’: 8, ‘key2’: ‘hello’, ‘key3’: [5,6,8]}

del x[‘key2’]

print(x)

=

{‘key1’: 8, ‘key3’: [5,6,8]}

* Loop through dictionary and print each key value pair on each line

x = {‘key1’: 8, ‘key2’: ‘hello’, ‘key3’: [5,6,8]}

for key, value in x.items():

print(key, value)

for key in x:

print(key, x.[key])

=

‘key1’ 8

‘key2’ ‘hello’

‘key3’ [5,6,8]

* Other methods for dictionaries

clear() Removes all the elements from the dictionary

copy() Returns a copy of the dictionary

fromkeys() Returns a dictionary with the specified keys and value

get() Returns the value of the specified key

items() Returns a list containing a tuple for each key value pair

keys() Returns a list containing the dictionary's keys

pop() Removes the element with the specified key

popitem() Removes the last inserted key-value pair

setdefault() Returns the value of the specified key. If the key does not exist: insert the key, with the specified value

update() Updates the dictionary with the specified key-value pairs

values() Returns a list of all the values in the dictionary

Functions

* Calling simple function

def function\_name():

print(‘123’)

function\_name()

=

123

* Calling function with arguments

def function\_name(x, y):

print(x, y)

print(x + y)

function\_name(5, 7)

=

5, 7

12

* Using return in a function(if you return multiple values it returns in a tuple)

def function\_name\_one(x, y):

return x \* y

def function\_name\_two(x, y):

return x \* y, x – y, x + y + y

print(function\_name\_one(5, 7))

print(function\_name\_two(5, 7))

=

35

(35, -2, 19)

* Returning multiple values from a function and avoiding they going into a tuple(instead into their own variable)

def function\_name(x, y):

return x \* y, x – y, x + y + y

sum\_var\_one, sum\_var\_two, sum\_var\_three = function\_name(5, 7)

print(sum\_var\_one, sum\_var\_two, sum\_var\_three)

=

35 -2 19

* Passing optional argument in a functions

def function\_name(x, y, z=None):

print(x, y, z)

function\_name(5, 7)

function\_name(5, 7, 99)

=

5, 7

5, 7, 99

* Anonymous function using lambda

Function\_var\_name = lambda x: x \* 4

print(function\_var\_name(5))

=

20

* Using map() and lambda function to edit a whole list of numbers

x = [1,3,4,2,3,8]

map\_var\_name = map(lambda i: i \* 2, x)

print(list(map\_var\_name))

=

[2,6,8,4,6,16]

* Using filter() and lambda function to filter a whole list of numbers, onling adding them to the new list if the number meets the lambda’s statement, meaning the statement equals true(in this case the statement will be only accepting even numbers)

x = [1,3,4,2,3,8]

filter\_var\_name = filter(lambda i: i % 2 == 0, x)

print(list(filter\_var\_name))

=

[4, 2, 8]

* Other built-in python functions that can be called

abs() Returns the absolute value of a number

all() Returns True if all items in an iterable object are true

any() Returns True if any item in an iterable object is true

ascii() Returns a readable version of an object. Replaces none-ascii characters with escape character

bin() Returns the binary version of a number

bool() Returns the boolean value of the specified object

bytearray() Returns an array of bytes

bytes() Returns a bytes object

callable() Returns True if the specified object is callable, otherwise False

chr() Returns a character from the specified Unicode code.

classmethod() Converts a method into a class method

compile() Returns the specified source as an object, ready to be executed

complex() Returns a complex number

delattr() Deletes the specified attribute (property or method) from the specified object

dict() Returns a dictionary (Array)

dir() Returns a list of the specified object's properties and methods

divmod() Returns the quotient and the remainder when argument1 is divided by argument2

enumerate() Takes a collection (e.g. a tuple) and returns it as an enumerate object

eval() Evaluates and executes an expression

exec() Executes the specified code (or object)

filter() Use a filter function to exclude items in an iterable object

float() Returns a floating point number

format() Formats a specified value

frozenset() Returns a frozenset object

getattr() Returns the value of the specified attribute (property or method)

globals() Returns the current global symbol table as a dictionary

hasattr() Returns True if the specified object has the specified attribute (property/method)

hash() Returns the hash value of a specified object

help() Executes the built-in help system

hex() Converts a number into a hexadecimal value

id() Returns the id of an object

input() Allowing user input

int() Returns an integer number

isinstance() Returns True if a specified object is an instance of a specified object

issubclass() Returns True if a specified class is a subclass of a specified object

iter() Returns an iterator object

len() Returns the length of an object

list() Returns a list

locals() Returns an updated dictionary of the current local symbol table

map() Returns the specified iterator with the specified function applied to each item

max() Returns the largest item in an iterable

memoryview() Returns a memory view object

min() Returns the smallest item in an iterable

next() Returns the next item in an iterable

object() Returns a new object

oct() Converts a number into an octal

open() Opens a file and returns a file object

ord() Convert an integer representing the Unicode of the specified character

pow() Returns the value of x to the power of y

print() Prints to the standard output device

property() Gets, sets, deletes a property

range() Returns a sequence of numbers, starting from 0 and increments by 1 (by default)

repr() Returns a readable version of an object

reversed() Returns a reversed iterator

round() Rounds a numbers

set() Returns a new set object

setattr() Sets an attribute (property/method) of an object

slice() Returns a slice object

sorted() Returns a sorted list

staticmethod() Converts a method into a static method

str() Returns a string object

sum() Sums the items of an iterator

super() Returns an object that represents the parent class

tuple() Returns a tuple

type() Returns the type of an object

vars() Returns the \_\_dict\_\_ property of an object

zip() Returns an iterator, from two or more iterators

\*args and \*\*kwargs

* Breaking tuples or lists into induvial elements using \*args

x = [7, 8, 8, 28, 15, 40, 4, 3, 9]

y = (7, 8, 8, 28, 15, 40, 4, 3, 9)

print(\*x)

print(\*y)

=

7 8 8 28 15 40 4 3 9

7 8 8 28 15 40 4 3 9

* Breaking dictionaries into induvial elements using \*\*kwargs

x = {‘key1’: 8, ‘key2’: ‘hello’, ‘key3’: [5,6,8]}

print(\*\*x)

=

8 ‘hello’ [5,6,8]

* Using \*args and \*kwarges as arguments in a function(it allows you to pass an endless amount of values in. \*args will be for elements and \*\*kwargs will be for key value pairs)

def function\_name(\*args, \*\*kwargs):

print(args, kwargs)

function\_name(1,3,4,5,6,4,7,key\_one=4,key\_two=87)

def function\_name(\*x):

print(\*x)

function\_name(1,3,4,5,6,4,7)

=

(1, 3, 4, 5, 6, 4, 7) {‘key\_one’: 4, ‘key\_two’: 87}

1 3 4 5 6 4 7

Handling Exceptions

* Stopping errors being thrown by using exceptions to catch them (stops whole program from crashing and stores the error in a var so you know what it is causing errors. If there is no error it’ll just continue unless there is a finial block like in the next example stating to respond otherwise.)

try:

x = 7 / 0

except Exception as var\_name\_storing\_exception:

print(var\_name\_storing\_exception)

=

x = 7 / 0

* Using a finally: block after an exception (it will always run no matter if a error was thrown during the exception or not. Often clean up code is put in here to catch any errors thrown in the exception to keep the code running and indicate the error to the programmer.)

try:

x = 7 / 0

except Exception as var\_exception:

finally:

if var\_exception == None:

print(‘no error’)

else:

print(f’Exception finished, error was found: {var\_exception}’)

=

x = 7 / 0