Python SPOT Wiki

Type coercions: explicit (e.g., using int(), str()) and implicit)

1: Which variable is coerced? Is it implicit or explicit coercion?

x = 3.5

y = 5

z = x + y

(The variable x is assigned to the float 3.5, the variable y is assigned to the integer 5. The variable z is assigned value of x + y which is a float, this is an example of implicit coercion)

2: What coercion is happening here? Is it implicit or explicit?

a = 1

b = 2

print(a + b)

(The value of the two integers added together is the integer 3. The print function coerces the 3 into a string, the print() function always coerces any argument into strings. This is an example of implicit coercion)

3: What coercion is happening here? Is it implicit or explicit?

month = "December"

day = int(input("What day is it? "))

print(f"Today is the {day} of {month}")

(The user is prompted for the day of the week. The value from the user is explicitly coerced into an integer with the int() function and assigned to the variable day. The print() function is called with an f-string prefix which allows string interpolation. Any expressions within the {} are replaced by the value of the expressions. The values are coerced implicitly into strings The concepts demonstrated here are string interpolation and both explicit and implicit coercion)

Numbers, including handling exceptions (ValueError, ZeroDivisionError)

Basic questions:

Are integers and floats mutable or immutable? (Immutable)

Are integers and floats primitive or non-primitive? (Primitive)

Are integers and floats literals? (Literals)

What is a literal? (Any syntatic notation that lets you directly represent an object in source code.

1: What does this return and why? What concept does this cover?

def convert\_to\_int(string):

try:

converted\_integer = int(string)

return converted\_integer

except ValueError:

return "That string cannot be converted to an integer"

print(convert\_to\_int("hello"))

(The defined function convert\_to\_int()takes one argument, a string and then explicitly coerces it into an integer. A try except block tests the value for a ValueError. If it passes it, the converted integer is returned. If it does not pass, then the warning string is returned. The concept here is implicit coercion, try except blocks,and ValueError’s.)

2: What does this return and why? What concept does this cover?

def division(number1, number2):

numerator = number1

denominator = number2

try:

result = numerator / denominator

return result

except ZeroDivisionError:

return "The denominator cannot be zero"

print(division(5, 0))

(The defined function division takes two integer parameters and divides the 1st number by the 2nd number. The try, except block tests the result for a ZeroDivisionError. If the mathematical value passes the test it is returned. If the value does not pass, then the string “The denominator cannot be zero” is returned. The arguments 5 and 0, when passed to the function division returns the warning string which is then printed. The concept demonstrated here is explicit coercion, try except blocks and ZeroDivisionError’s)

3: What does this print and why, what concept does this demonstrate?

def addition(number1, number2):

number1 += number2

x = 1

y = 2

addition(x, y)

print(f"x is {x}, y is {y}")

The defined function addition takes two parameters, both integers and then adds the two integers together using the += shortcut operator. The function addition does not explicitly return any values. Because integers are immutable, the function addition does not modify the variable x as expected. This prints x is 1, y is 2. The concept here is the immutability of integers.)

4. What does this print and why? What concept does this cover? How would you refactor this to remove the space?

print(2 + 3 \* 4, 4 \* (3 + 2))

(This will output `14 20` to the screen. Integers can be added together within a print function. The comma allows multiple different mathematical calculations to occur within the same print() function call. This To remove the space in the output, use the optional sep= parameter with and empty space ’ ‘. print(2 + 3 \* 4, 4 \* (3 + 2), sep=' ') The main concept here is order of operations, and the use of commas to allow for separate mathematical operations within a print() function.)

5. What can be used in place of commas to make this more readable?

print(123112940)

(Underscore can replace commas print(123\_112\_940))

Strings

Basic questions:

Are strings mutable or immutable? (Immutable)

Are strings primitive or non-primitive? (Primitive)

Are strings literals? (Literals)

What is a text sequence? (strings of characters)

What kind of characters are used in a string? (Unicode characters)

Are text sequences the same as ordinary sequences? (No, text sequence only

contain any object, only characters.)

1. What is the output of this code, and why? What is the concept covered here?

str1 = "Hello, world!"

sub1 = str1[8:12]

print(sub1)

sub2 = str1[::-1]

print(sub2)

sub3 = str1[::2]

print(sub3)

(The concept is string splicing, which creates a new string object, a substring, from the original string object based on the indexing sequence given and the step splice argument if any. sub1 is returning the characters from and inclusive of index 8 to but exclusive of index 12, or the string world, which is then printed. The substring sub2 returns the entire string, but the slice step argument of -1 reverses the order of the indexing returning the string reversed, or !dlrow, olleH, which is then printed. The last substring returns the entire original string, but the slice step argument of 2 instructs the indexing to skip every other index starting with 1. The new substring has a value of Hlo ol! which is printed.)

2. What does this print and why? What concept is this?

print(“Hello\nWorld”)

(This demonstrates the escape clause \. This will print the string Hello and then create a new line and print World)

3. What does this print and why? What concept is this?

name = ‘Alexander Graham Bell’

print(name[0])

(This will print the character A and it demonstrates accessing strings with indexing syntax.)

f-strings

Basic Questions:

What are f-strings? (string prefix for defining formatted string literals that enables string interpolation)

1. What does this print and why, what is the concept?

name = ‘Abraham Lincoln’

print(f’”{name} was a President of the US”)

(This will print Abraham Lincoln was a President of the US. The f prefix in the print function allows the value of the variable to be interpolated and merged with the rest of the print statement. This demonstrates f-strings.)

string methods

Basic Questions:

How do you identify a method versus a function?

(A method occurs when an object is followed by a . and then followed by a function invocation. Whereas a function is a function invocation followed by () with the object passed into the ().

1. What does this print and why?

mashup = “thIs is How we type careLEssly”

cleaned = mashup.capitalize()

print(cleaned)

2. What do these print and why?

stuff = ‘tHIS iS bACKWARDS’

str1 = stuff.swapcase()

str2 = stuff.upper()

str3 = stuff.lower()

print(stuff)

print(str1)

print(str2)

print(str3)

3. What do these print and why?

s1 = "Hello"

print(s1.isalpha())

s2 = "Hello World"

print(s2.isalpha())

s3 = "Hello!"

print(s3.isalpha())

s4 = "Hello123"

print(s4.isalpha())

s5 = ""

print(s5.isalpha())

s6 = "こんにちは"

print(s6.isalpha())

s7 = "HelloWorld"

print(s7.isalpha())

words = ["apple", "banana", "cherry"]

print(all(word.isalpha() for word in words))

4. What does this print and why?

string1 = "HelloWorld"

string2 = "12345"

string3 = "Hello World"

result1 = string1.isalpha()

result2 = string2.isalpha()

result3 = string3.isalpha()

print("Is '{}' alphabetic?".format(string1), result1)

print("Is '{}' alphabetic?".format(string2), result2)

print("Is '{}' alphabetic?".format(string3), result3)

5. What do these print and why?

s1 = "123abc"

print(s1.isdigit())

s2 = "123$%^"

print(s2.isdigit())

s3 = ""

print(s3.isdigit())

s4 = "12345"

print(s4.isdigit())

6. What do these print and why?

print("Hello World".isalnum())

print("Hello@World".isalnum())

print("".isalnum())

print("Hello123".isalnum())

7. What do these print and why?

name = 'HELLO'

if name.isupper():

print("WORLD")

else:

print("world")

8. What do these print and why?

def punctuation\_type(str):

if str.upper():

print('This is all caps')

elif str.lower():

print('This is all lowercase')

else:

print('Neither')

str1 = 'HELLO'

str2 = 'yolo'

str3 = 'My Name Is: '

punctuation\_type(str1)

punctuation\_type(str2)

punctuation\_type(str3)

9. What do these print and why?

str1 = " "

str2 = " Hello "

str3 = "Hello World"

print(str1.isspace())

print(str2.isspace())

print(str3.isspace())

sentence = "Hello World! How are you? "

word\_count = sum(1 for word in sentence.split() if word.isspace())

print("Number of words in the sentence:", word\_count)

10. What do these print and why?

s = " Hello, World! "

print(s.strip())

print(s.strip(" !"))

11. What do these print and why?

s = "www.example.com"

print(s.lstrip('wcmo.'))

12. What do these print and why?

s = 'impatient'

print(s.rstrip('tp'))

print(s.rstrip('p'))

13. What do these print and why?

s = "Hello, World!"

print(s.replace("Hello", "Hi"))

print(s.replace("o", "0"))

14. What do these print and why?

sentence = "This is a sample sentence."

words = sentence.split()

print(words)

csv\_data = "John,Doe,30,New York"

fields = csv\_data.split(",")

print(fields)

sentence = "This is a sample sentence."

words = sentence.split(maxsplit=2)

print(words)

15. What does this print and why?

str1 = "hello world"

str2 = str1.capitalize()

print("Original string:", str1)

print("Capitalized string:", str2)

boolean vs. truthiness

Basic Question:

In Python, what values are considered Falsy and what are considered Truthy?

1. What do these print and why?

truthy\_values = [1, 2, 3, "hello", [1, 2, 3], {"a": 1}, True, 0, "", [], {}, None, False]

print(“Values:”)

for value in truthy\_values:

if value:

print(f"{value} is truthy")

else:

print(f"{value} is falsy")

2. What do these print and why?

x = 5

y = 10

z = 15

print(x > 0 and y < 20)

print(not x == y)

print(x < y and y < z)

print(x > y or y > z)

print(not (x > y))

3. What do these print and why?

a = 10

b = 20

print(a < b < 30)

print(a > b or b == 20)

4. What do these print and why?

my\_list = [1, 2, 3, 4, 5]

print(3 in my\_list)

print(6 not in my\_list)

5. What do these print and why?

temperature = 25

time\_of\_day = "morning"

if temperature < 30 and (time\_of\_day == "morning" or time\_of\_day == "afternoon"):

print("It's a pleasant day!")

else:

print("It's either too hot or not the right time of day.")

6. What does this print and why?

num = 12

if num / 3 < 3 and num > 10:

print("Hello") # False

elif num >= 8 and num < 6 or num > 4 and num < 16:

print("Hello 2")

elif num % 4 == 0 or num < 7 and num < 10:

print("Hello 3")

else:

print("Buy")

ranges

Basic questions:

Is a range primitive or non-primitive?

Is a range mutable or immutable?

Does range have a literal form or a type constructor?

Is a range a sequence or a collection?

What is the most common use of the range datatype?

Are ranges homogenous or heterogeneous?

Why are ranges considered lazy?

1. What do these print and why? What concept does this demonstrate?

print(range(0,10)

print(len(range(5, 15)))

print(my\_range[1])

print(str(range(3, 7)))

print(list(range(12, 8, -1)))

print(list(range(5, 5, 1)))

print(5 in range(5))

print(5 not in range(5, 10))

2. What does this code print and why? What concept does this demonstrate?

example = range(0)

if example:

print(list(example))

else:

print(example)

3. What does this code print and why? What concept does this demonstrate?

def number\_range(number):

match number:

case n if n < 0:

print(f'{number} is less than 0')

case n if n <= 50:

print(f'{number} is between 0 and 50')

case n if n <= 100:

print(f'{number} is between 51 and 100')

case \_:

print(f'{number} is greater than 100')

number\_range(0)

number\_range(25)

list and dictionary syntax

Basic Questions

What categories are lists and dictionaries?

Are they mutable or immutable?

Are they primitive or non-primitve?

Are they literals, or do they require type constructors?

Are they sequences?

Does the order of the elements in both matter?

list methods: len(list), list.append(), list.pop(), list.reverse()

1. What does this print and why?

my\_list = [1, 2, 3, 4, 5]

length\_of\_list = len(my\_list)

print("Length of the list:", length\_of\_list)

2. What does this print and why?

lst\_one = [0, 1, 2, 3]

lst\_two = lst\_one.append(4)

if lst\_two:

print(lst\_two)

else:

print(lst\_one)

3. What does this print and why?

my\_list = [1, 2, 3, 4, 5]

ele = my\_list.pop()

print("Popped element:", ele)

print("List after popping:", my\_list)

ele1 = my\_list.pop(1)

print("Popped element at index 1:", ele1)

print("Modified list after popping at index 1:", my\_list)

4. What does this print and why?

elements = [0, 1 , 2, "Dima"]

print(elements.reverse())

print(elements)

5. What does this print and why?

ages = {

"dimo": 31,

"olena": 32,

"tetiana": 28

}

def get\_val\_of\_dima(info):

try:

info['dimo']

return info['dimo']

except KeyError:

return "Typo"

print(get\_val\_of\_dima(ages))

6. What does this print and why?

my\_dict = {'a': 1, 'b': 2, 'c': 3}

keys = my\_dict.keys()

print(keys)

for key in keys:

print(key)

7. What does this print and why?

my\_dict = {'a': 1, 'b': 2, 'c': 3}

values = my\_dict.values()

print(values)

for value in values:

print(value)

8. What does this print and why?

my\_dict = {'a': 1, 'b': 2, 'c': 3}

items = my\_dict.items()

print(items)

for key, value in items:

print( key, value)

variable scope, global keyword, variables as pointers, variable shadowing

1. What does this print and why?

name = 'John'

def greet():

print(f"Hello, {name}!")

greet() # Output: Hello, John!

2. What does this print and why?

def assign():

var = 20

print(var)

assign()

try:

print(var)

except NameError as e:

print("Error occurred")

3. What does this print and why?

var = 10

def function1():

var = 20

print(var)

function1()

try:

print(var)

except NameError:

print("Error occurred")

def function2():

var += 5

print(var)

try:

function2()

except UnboundLocalError:

print("Error occurred")

def function3():

global var

var += 5

print(var)

function3()

print(var)

4. What does this print and why?

var = 10

def function1():

print(var)

function1()

def function2():

var = 20

print(var)

function2()

print(var)

5. What does this print and why?

def function1():

x = 10

def function2():

y = 20

print(x)

function2()

print(x)

function1()

print(x)

print(y)

6. What does this print and why?

var = 10

def access():

print(var)

access()

def modify():

global var

var += 5

` print(var)

modify()

print(var)

7. What does this print and why?

list1 = [1, 2, 3]

list2 = list1

list1.append(4)

print("list1:", list1)

print("list2:", list2)

8. What does this print and why?

x = 5

y = x

x = 10

print("x:", x)

print("y:", y)

9. What does this print and why?

x = 5

def function():

x = 10

print(x)

function()

print(x)

10. What does this print and why?

x = 5

def function(x):

print(x)

function(10)

print(x)