When you multiply a string by an integer in Python, it creates a new string by repeating the original string by the given number of times. For example, `'hello' * 3` will create a new string `'hellohellohello'`.

→ see bottom

When you multiply by 0, nothing will be printed

Check of criteria kloppen , not not wordt weer true, dus functies kloppen, maar not zegt van niet dus false maar staat nog een keer not

Assume you have the following variables:

```
x = 'abc'
y = 3
z = 2.0
```

What will be printed after these lines of code?

```
print(x*y)
print(x*z)
print(y*z)
```

```
'abcabcabc'
```

The code will result in a `TypeError` because you can't multiply a float and a string together.

```
6.0
```

```
x = 1
y = 15
print(not(not(x < 3) and not(y > 14 or y > 10)))
```

```
def outer():
    a = 5
    def inner():
        a = 10
        print("Value of a inside inner():", a)
    inner()
    print("Value of a inside outer():", a)
outer()
```

=

C

```
def outer():
    x = 10
    def inner():
        global x
        x = 5
    inner()
    print("Value of x inside outer():", x)
outer()
print("Value of x outside outer():", x)
```

```
x = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
y = {1: 50, 2: 40, 3: 30, 4: 20, 5: 10}
z = x.copy()

for key, value in x.items():
    if key in y:
        if value <= y[key]:
            z[key] = y[key]
        z[key] = y[key]

print(z)
```

```
{1: 50, 2: 40, 3: 30, 4: 20, 5: 10}
```

Volledig y omdat laatste regel erbij is gekomen

```
sample_dict = {
    "name": "Kim",
    "age": 25,
    "birthdate": "3-2-1998",
    "city": "Amsterdam"}
```

Suppose you want to create a dictionary that only contains the keys 'name' and 'city'.

```
keys = ["name", "city"]
new_dict = {}
for k in keys:
    new_dict[k] = sample_dict[k]
new_dict

keys = ["age", "birthdate"]
new_dict = {}
new_keys = sample_dict.keys() - keys
for k in new_keys:
    new_dict[k] = sample_dict[k]
new_dict
```

```
closet = {
    "shirts" : 5,
    "colors" : ['red', 'yellow', 'blue', 'pink']
}

closet["shoes"] = ["sneakers", "heels"]
closet["shirts"] += 2
```

So the final dictionary should look like:

```
{'shirts': 7, 'colors': ['red', 'yellow', 'blue', 'pink'], 'shoes': ['sneakers', 'heels']
```

```
list1 = [1, 2, 3, 4, 5]
list2 = [2, 4, 6, 8, 10]
list3 = []

for i in list1:
    for j in list2:
        if i + j >= 10:
            break
        list3.append(i + j)

print(list3)
```

A `[3, 5, 7, 9, 4, 6, 8, 5, 7, 9, 6, 8, 7, 9]`

Suppose you have the following function:

```
def add_numbers(num1, num2 = 10, num3 = 20):
    return num1 + num2 + num3
```

What are the outputs if we call the function three times as follows:

```
add_numbers(5, 15)
add_numbers(5, num3 = 30)
add_numbers(num2 = 15, num3 = 10)
```

```
40
45
```

The code will result in `TypeError` because there is a missing argument.

Suppose you have the following functions, one to elevates a number to square and the second elevates to cube:

```
def square(n):
    return (n**2)
def cube(n):
    return (n**3)
```

You want to apply both functions to the elements in a list at the same time.

For example:

If you have a list such as:
`my_list = [0, 1, 2, 3, 4]`

you want to print:

```
[0, 0]
[1, 1]
[4, 8]
[9, 27]
[16, 64]
```

Which of the following blocks of codes works as intended?

```
funcs = [square, cube]

for i in my_list:
    results = map(lambda x: x(i), funcs)
    print(list(results))
```

Assume you already have a variable `a`, which is an integer between 1 and 4.

What is the output of the following code snippet?

```
mylist = [1, 3, 5, 7, 9]
print(mylist[a] + mylist[-a])
```

A 12

Starts at 0, so a1 = second number, this does not apply to -a

Consider the following code snippet:

```
mylist = [1, 3, 2, 3, 4, 5, 3, 3]
mylist.append('3')
mylist.remove(3)
print(mylist.count(3))
```

Append ads value as a string, so leaves 3 in list and 1 in string. Mylist.counts values in list, so 3.

What will be the output?

Hint: The `remove()` list method removes the first occurrence of the element with the specified value.

A 3

```
def income_tax_calculator(gross_income):

    results = {}

    rate_1 = 0.10
    rate_2 = 0.12
    rate_3 = 0.22
    rate_4 = 0.32

    bracket_1 = 11000
    bracket_2 = 45000 - 11000
    bracket_3 = 95000 - 45000

    if gross_income > 95000:
        total_tax = ((gross_income - 95000) * rate_4) + (bracket_3 * rate_3) + (bracket_2
+ (bracket_1 * rate_1)
    elif 95000 >= gross_income > 45000:
        total_tax = ((gross_income - 45000) * rate_3) + (bracket_2 * rate_2) + (bracket_1
    elif 45000 >= gross_income > 11000:
        total_tax = ((gross_income - 11000) * rate_2) + (bracket_1 * rate_1)
    else:
        total_tax = gross_income * rate_1

    net_income = gross_income - total_tax

    results['total_tax'] = round(total_tax, 2)
    results['net_income'] = round(net_income, 2)

    return results
```

You need to write a function called `print_info` which takes two required arguments called `name` and `age`, and a flexible number of keyword arguments.

The function should return a dictionary with `name`, `age`, and all other key-value pairs that may be passed as keyword arguments.

For example:

If we call the function as:
`print_info('John', 30, city = 'New York', job = 'software engineer')`

then it should return the dictionary:
`{'name' = 'John', 'age' = 30, 'city' = 'New York', 'job' = 'software engineer'}`

Which of the following code snippets will do what you want?

```
def print_info(name, age, **kwargs):
    info = {}
    info['name'] = name
    info['age'] = age
    for key, value in kwargs.items():
        info[key] = value
    return info
```

It is important to add key values

```
x = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
y = {1: 50, 2: 40, 3: 30, 4: 20, 5: 10}
z = x.copy()
for key, value in x.items():
    if key in y:
        if value <= y[key]:
            z[key] =  y[key]
print(z)
```

```
{1: 50, 2: 40, 3: 30, 4: 40, 5: 50}
```

Assume you already have a variable called alphabet, which contains a string, e.g., `alphabet = "abcdefghijklmnopqrstuvwxyz"`

Which of the following code snippets will give the same output as the following command?
`print(list(alphabet[1::2]))`

Hint: The `reverse()` method reverses the sorting order of the elements in a list.

A

```
values = []
for index, value in enumerate(alphabet, start=1):
    if index % 2 == 0:
        values.append(value)
print(values)
```

You need to write a function called `sort_list` that accepts a list of integers.

The function should return a new list, in which the elements are sorted from highest to lowest based on their absolute values.

For example:
If we call the function as:
`sort_list([-9, 2, 5, -3, -10, 4, 7])`

then it should return the list:
`[-10, -9, 7, 5, 4, -3, 2]`

Which of the following functions will give you the correct output?

```
def sort_list(x):
    return sorted(x, key = lambda num: abs(num), reverse = True)
```

Take the following variables:

```
a = '1'
b = 0
```

What will be printed after this line of code?

```
print(a * b)
```

When you multiply a string by an integer in Python, it creates a new string by repeating the original string by the given number of times. For example, `'hello' * 3` will create a new string `'hellohellohello'`. → see bottom

When you multiply by 0, nothing will be printed

float =2.0 integer = 1 string = ‘1’ → eerst geldende functie klopt

You need to write a function called **main** that takes a list of strings as an argument. The function should return a dictionary, in which the keys are the strings from the input list and the values are integers representing the length of the strings.

For example, the return value from:

```
main(['a', 'ab', 'abc', 'abcd'])
```

should be:

```
{'a': 1, 'ab': 2, 'abc': 3, 'abcd': 4}
```

Which of the following function definitions would work as intended?

A
```
def main(x):
    y = {}
    for i in x:
        y[i] = len(i)
    return y
```

Take the following variables:

```
a = '1'
b = 0
```

What will be printed after this line of code?

```
print(a + b)
```

Cannot add string and integer diretly

x and **y** contain dictionaries. Some, but not all, of the keys in **x** also exist in **y**. You need to write a program that creates a new dictionary **z** that has all the items of **x**, but for the keys that exist in both **x** and **y**, such items have the values of the items in **y**.

For example, if:

```
x = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
y = {1: 100, 'a': 'AAA', 2: 200, 'b': 'BBB', 3: 300, 'c': 'CCC'}
```

then the new dictionary should be:

```
z = {1: 100, 2: 200, 3: 300, 4: 40, 5: 50}
```

A
```
z = x.copy()
for i in x:
    if i in y:
        z[i] = y[i]
```

What is the output of the following code?

```
list_1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(list_1[1] + list_1[-1])
```

11

x and **y** are dictionaries that have the same keys, but different values. You need to write a program that creates a new dictionary **z** with the same keys as **x** and **y**. The values should be two-element lists that contain first the corresponding value in **x**, then the corresponding value in **y**.

For example, if:

```
x = {1: 10, 2: 20, 3: 30}
y = {1: 15, 2: 25, 3: 35}
```

```
z = {}
for key, value in x.items():
    z[key] = [value + y[key], value - y[key]]
```

x.items mergers the key value pairs

then the new dictionary should be:

```
z = {1: [10, 15], 2: [20, 25], 3: [30, 35]}
```

Which of the following programs will work as intended?

A
```
z = {}
for key, value in x.items():
    z[key] = [value, y[key]]
```

```
x = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
y = {1: 100, 20: 'AAA', 3: 300, 40: 'BBB', 5: 500}
x.update(y)
print(x)
```

```
{1: 100, 2: 20, 3: 300, 4: 40, 5: 500, 20: 'AAA', 40: 'BBB'}
```

How do you extract items from a dictionary to a list of tuples, where the first item of a tuple is a key from the original dictionary and the second item of a tuple is the value associated with said key?

For example, if:

```
x = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
```

your list should be:

```
y = [(1, 10), (2, 20), (3, 30), (4, 40), (5, 50)]
```

```
y = list(x.items())
```

```
y = []
for i in x.keys():
    y.append((i, x[i]))
```

You have the following function:

```
def multiplier(n, x=2):
    return n * x
```

What is the return value from the following function call?

```
multiplier('3.0')
```

‘3.03.0’

```
x = ["pentagon", 'hexagon', 'heptagon', 'octagon', 'nonagon', 'decagon']
y = {5: 'pentagon', 6: 'hexagon', 7: 'heptagon', 8: 'octagon', 9: 'nonagon', 10: 'dec
```

Which of the following program creates **y** as intended?

A
```
y = dict(enumerate(x, start=5))
```

You need to write a program that gives the natural logarithms of each integer in the list **x**, where:

```
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Python does not have a built-in function to calculate the natural logarithms, but you can use a function called **log** from the **numpy** package.

Which of the programs below will give you what you need?

```
from math import log list(map(lambda i: log(i), x))
```

A
```
import numpy as np
list(map(lambda i: np.log(i), x))
```

You need to write a function called **main** which takes a list of integers as an argument. The function should return the square of the sum of all even numbers.

For example:

```
main([1, 2, 3, 4, 5])
```

should return:

```
36
```

```
def main(x):
    total = 0
    for i in x:
        if i % 2 == 0:
            total += i
    return total**2
```

```
def main(x):
    total = 0
    for i in x:
        if i % 2 == 1:
            continue
        total += i**2
    return total
```

```
x = ['First', 'Second', 'Third']
y = [(1, 'First'), (2, 'Second'), (3, 'Third')]
```

```
y = list(enumerate(x, start=1))
```

In the following code, **x** has an integer value: After the following code

```
a = x // 3
b = x % 3
outcome = a * 3 + b
```

What will be the value of **outcome**?

Same as x

Let's define three variables:

```
a = 1
b = 2.0
c = '3'
```

What is the value of `result` at the end of the following code?

```
result = 0
if type(a) == type(0) and type(a) == type(b) and type(a) == type(c):
    result = a
if type(b) == type(0) or type(b) == type(c):
    result = b
if type(c) != type(0):
    result = c
```

‘3’

Volledige funtctie moet kloppen dan = 0 anders c. het gaat erom welke functie houd

You already have a list named **list_1** with an even number of integers. The following few lines of code print a number to the screen:

```
result = 0
for index, value in enumerate(list_1):
    if index % 2 == 0:
        result += value
print(result)
```

Which of the following four code snippets would print the same number?

```
print(sum(list_1[::2]))
```

```
def adder(n, x=2):
    return n + x
```

What is the return value from the following function call?

```
adder('3.0')
```

Cannot add string and integer

```
list_1 = [1, 3, 7, 4, 9, 2, 6, 8, 10]
```

One of the following code lines prints `9` as an output. Which one?

print(len(list_1))

Let's define three variables:

```
a = 1
b = 2.0
c = '3'
```

What is the value of **`result`** at the end of the following code?

```
result = 0
if type(a) == type('0') and type(a) == type(b) and type(a) == type(c):
    result = a
if type(b) == type('0') or type(b) == type(c):
    result = b
if type(c) != type('0'):
    result = c
```

0

In the following code, **x** and **y** have positive integer values:

```
outcome = x // y - int(x/y)
```

Always 0

You already have a list named **list_1** with an even number of integers. The following few lines of code print a number to the screen:

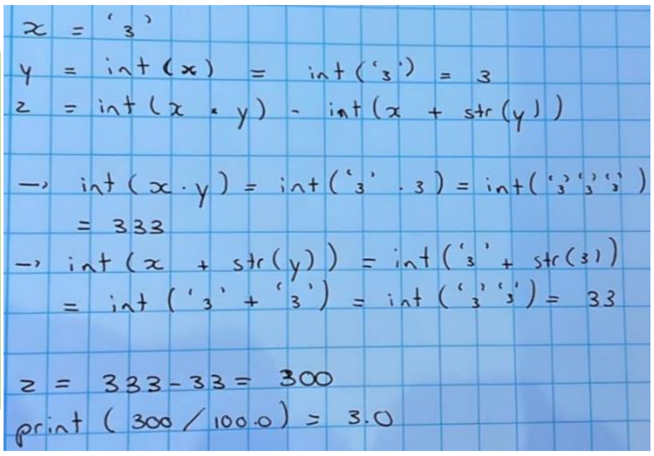
```
result = 0
for index, value in enumerate(list_1):
    if index % 2 > 0:
        result += value
print(result)
```

=

e following four code snippets would print the same number?

```
result = 0
for index, value in enumerate(list_1):

result += value if index % 2 == 1 else 0 print(result)
```



x = ‘3’

y = int(x) = int(‘3’) = 3

z = int(x * y) - int(x + str(y))

→ int(x · y) = int(‘3’ · 3) = int(‘3’‘3’‘3’)

= 333

→ int(x + str(y)) = int(‘3’ + str(3))

= int(‘3’ + ‘3’) = int(‘3’‘3’) = 33

z = 333 - 33 = 300

print(300 / 100.0) = 3.0

Take the following variables:

```
a = '1'
b = 0
```

What will be printed after this line of code?

```
print(a * b)
```