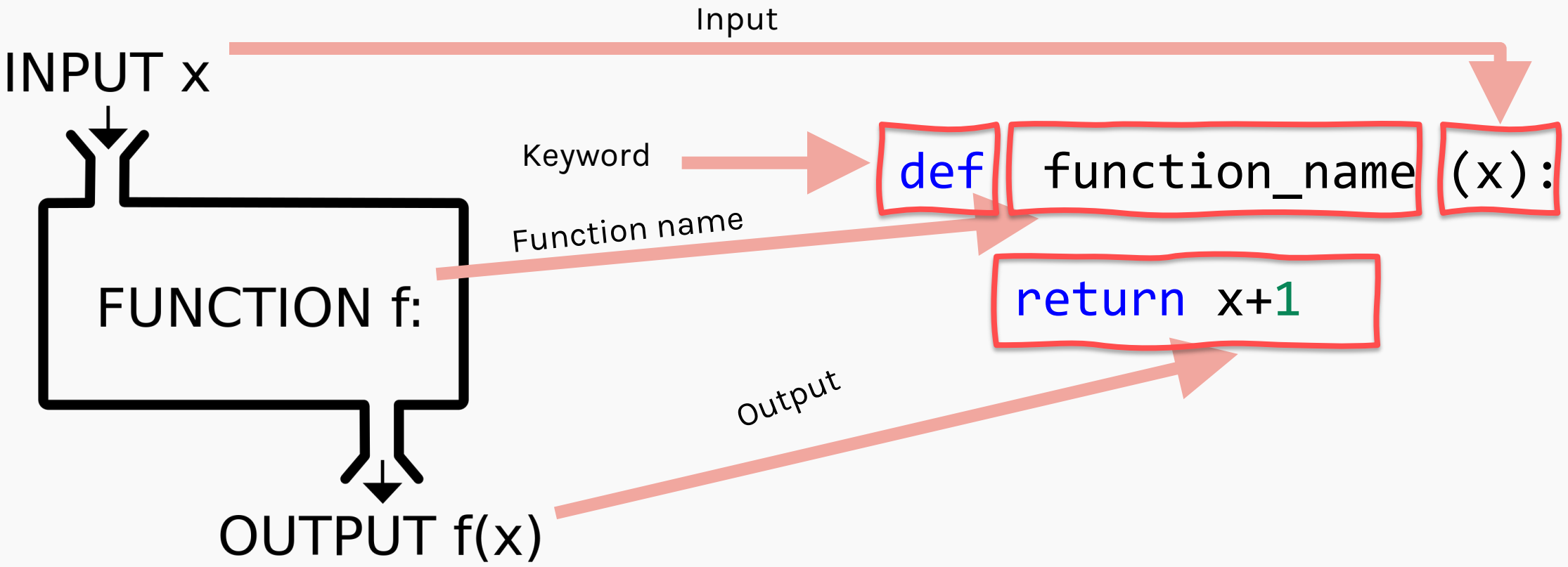
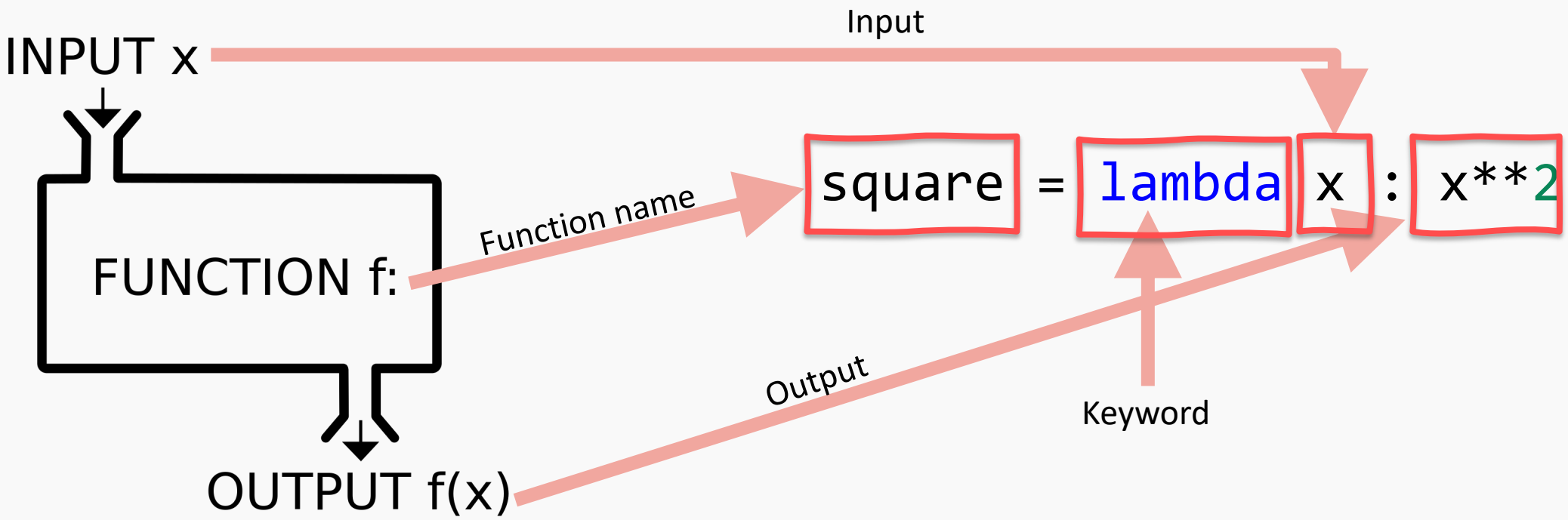
Lambda functions

STANDARD WAY OF DEFINING PYTHON FUNCTIONS



Lambda function



Lambda function

```
(SYNTAX) square = lambda x : x**2
```

- Lambda functions (also called **Anonymous functions**) are a concise way to declare functions.
- This pythonic way of defining functions avoids the keyword **RETURN**
- Very helpful when working with mathematical function implementation.

```
In [89]: square = lambda x: x**2


In [90]: square(2)
Out[90]: 4


In [91]: square(2,2)
---------------------------------------------------------------------------
TypeError                                 Trace
back (most recent call last)
<ipython-input-91-fc957e4c04b0> in <module>
----> 1 square(2,2)


TypeError: <lambda>() takes 1 positional
argument but 2 were given
```

Use lambda functions

The list.sort() and sorted() methods can take lambda functions

```
#Example of Sort function(sort in age and in potential)
lst = [('De Ligt',19,92.1), ('Alexander-Arnold',20,90.5),
       ('Sancho',19,94.0),('Havertz',20,93.2),('Rodrygo',18,91.4)]

lst.sort(key=lambda x:x[1]) #Sort by age
print(lst)
lst.sort(key=lambda x:x[2]) #Sort by potential
print(lst)
>>>
[('Rodrygo', 18, 91.4), ('De Ligt', 19, 92.1), ('Sancho', 19, 94.0), ('Alexander-Arnold',
20, 90.5), ('Havertz', 20, 93.2)]

[('Alexander-Arnold', 20, 90.5), ('Rodrygo', 18, 91.4), ('De Ligt', 19, 92.1), ('Havertz',
20, 93.2), ('Sancho', 19, 94.0)]
```

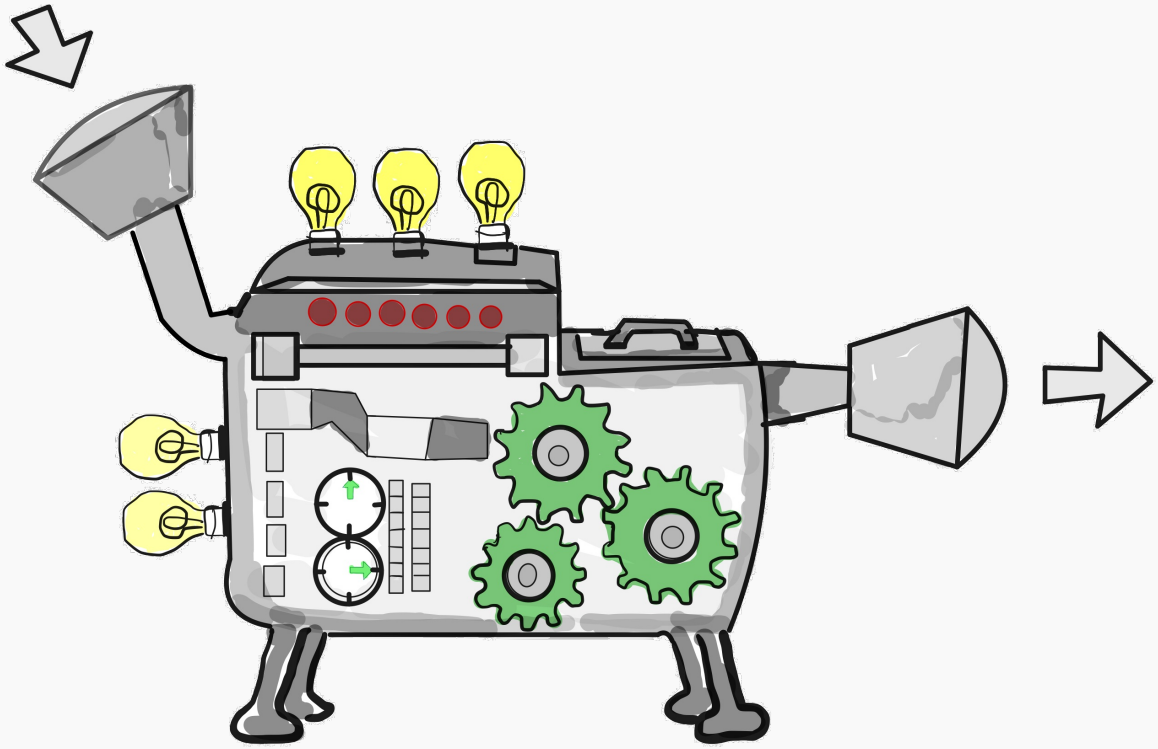
It is NOT compulsory to use a lambda function. Any regular function would do as well

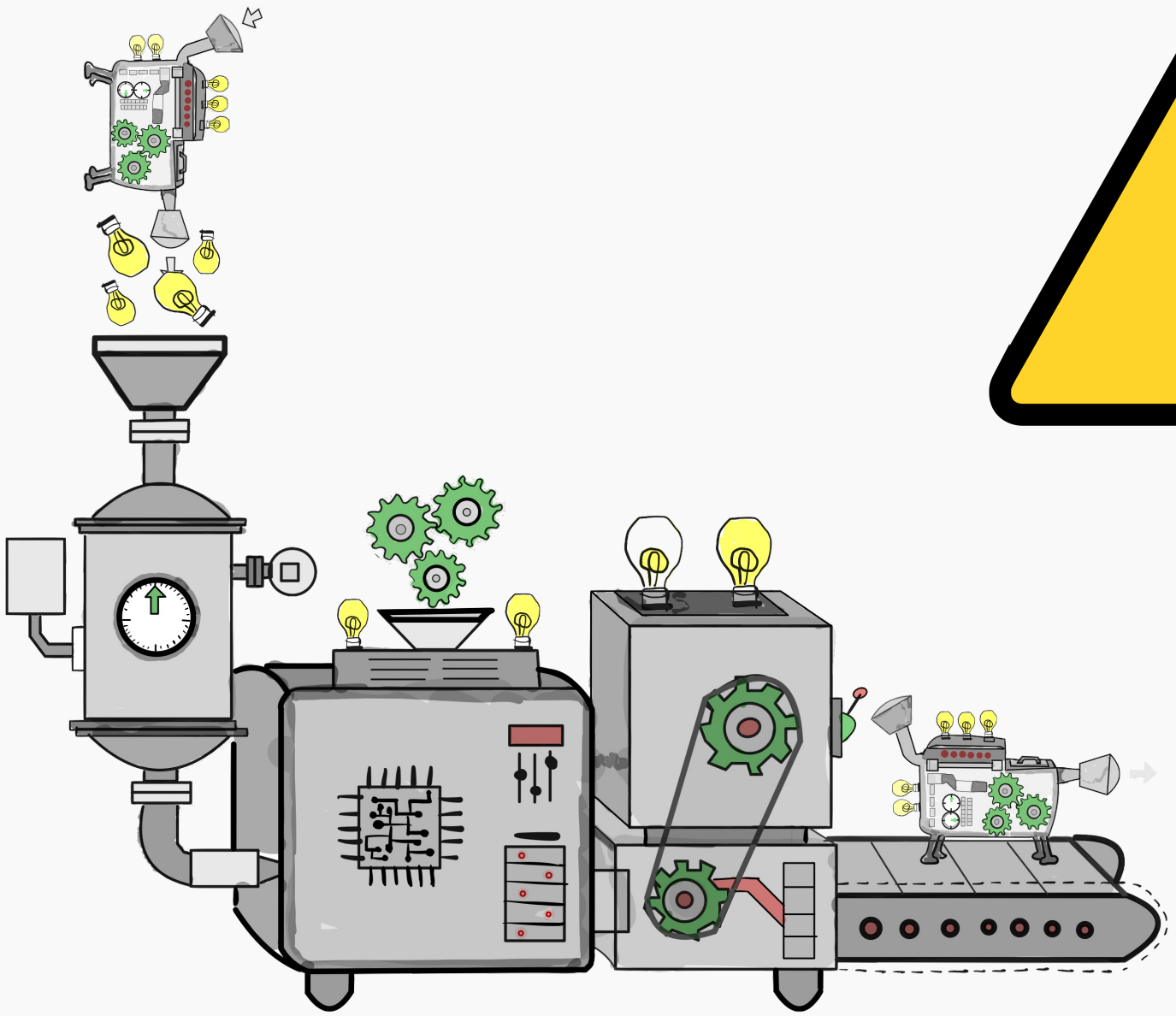
Use lambda functions

- Certain methods/functions exist which make it very easy to apply lambda functions on elements of iterable objects.
- map(), filter() both take a function as an input. (see e.g. below)

```
## Usage of lambda functions

#Example of Map function(calculate discount)
a = list(map(lambda x: x - 0.2*x, [120,300,150,3000,42.55,67.11]))

#Example of Filter function(get all 'back' positions)
b = list(filter(lambda x: 'B' in x, ['LB', 'CB', 'ST','CAM','CMD','GK','RB']))

print(a)
print(b)
>>>
[96.0, 240.0, 120.0, 2400.0, 34.04, 53.688]
['LB', 'CB', 'RB']
```

Advanced Functions

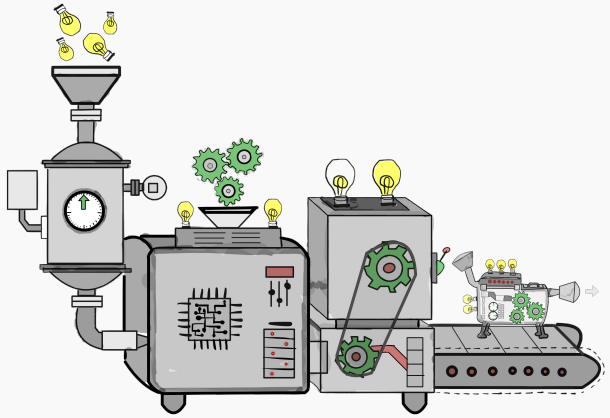




Advanced functions

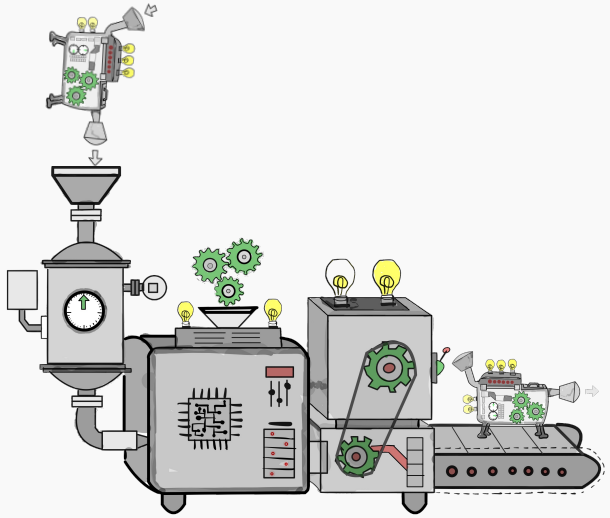
Level #1

Number in, Function out



Level #2

Function in, Function out



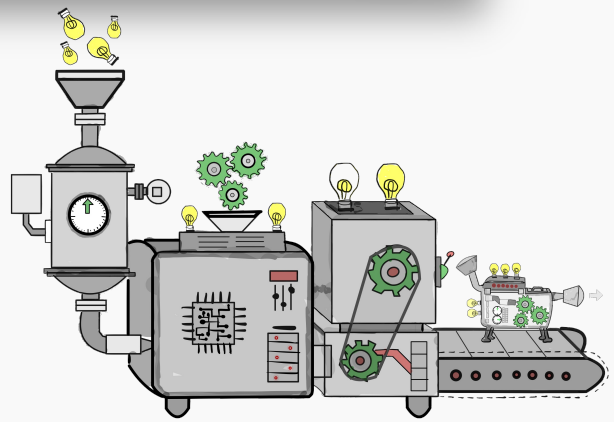
Level #1 - Number in, function out

- Combine a lambda function and use it inside a function to declare an even higher level of generalization.
- It means you are now returning a function, the specifics of which are decided by the input to higher function.

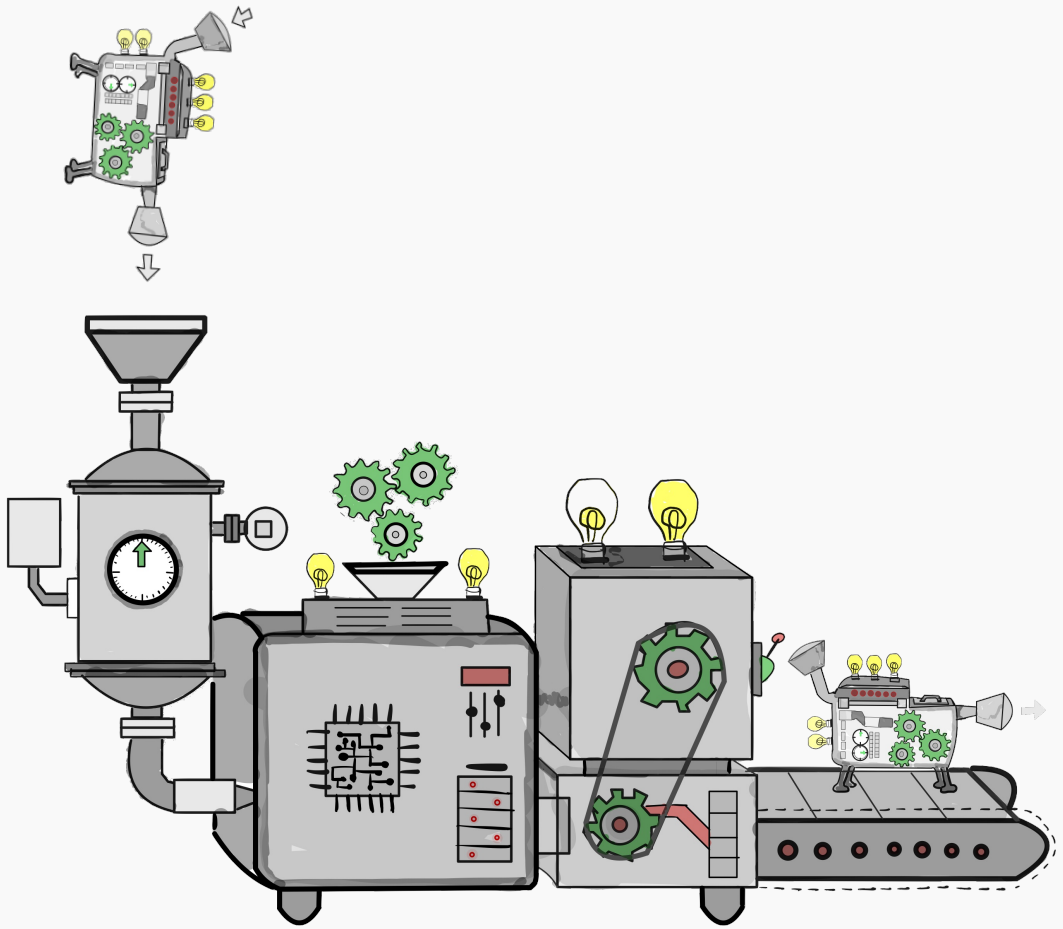
You are returning a function here

```
def multiplier(n):
    func = lambda x:x*n
    return func

doubler = multiplier(2)
tripler = multiplier(3)

print(doubler(100))
print(tripler(100))
>>>
200
300
```

It is an n-multiplier function. Based on input n the output function will vary



Level #2: Function in, Function out



Level #2 – Function in, Function out

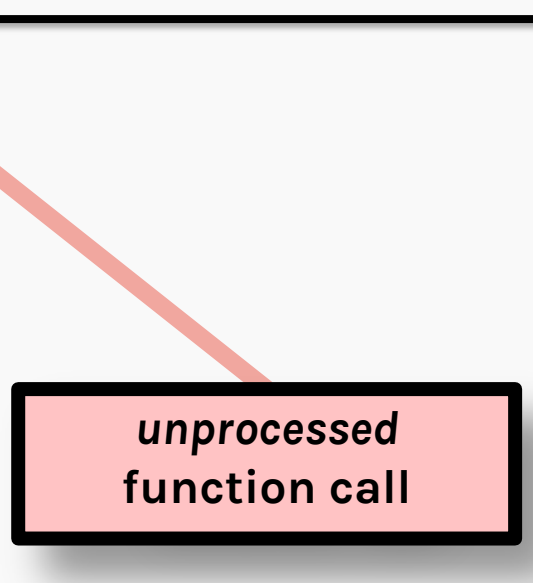
```
In [19]: def decorator_function(original_function):
    ...:     def packaging(x):
    ...:         print("Pavlos's function. All rights reserved")
    ...:         return original_function(x)
    ...:     return packaging
```

Taking a function as input

You are returning a function here

Level #2 – Function in, Function out

```
In [19]: def decorator_function(original_function):
    ...:     def packaging(x):
    ...:         print("Pavlos's function. All rights reserved")
    ...:         return original_function(x)
    ...:     return packaging

In [21]: def square(x):
    ...:     return x**2

In [27]: square(5)
Out[27]: 25
```



unprocessed
function call

Level #2 – Function in, Function out

```
In [19]: def decorator_function(original_function):
    ...:     def packaging(x):
    ...:         print("Pavlos's function. All rights reserved")
    ...:         return original_function(x)
    ...:     return packaging

In [21]: def square(x):
    ...:     return x**2

In [27]: square(5)
Out[27]: 25

In [22]: processed_square = decorator_function(square)
```

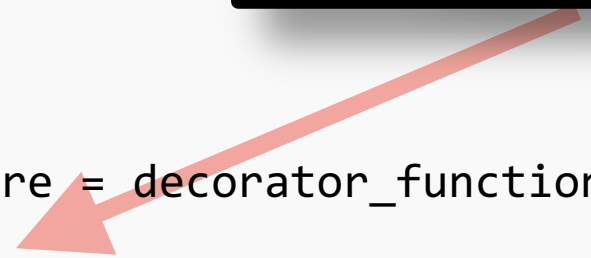
Remember!
decorator_function()
returns a function

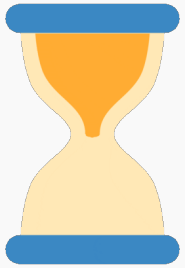
after processing using the decorator function

Level #2 – Function in, Function out

```
In [19]: def decorator_function(original_function):
    ...:     def packaging(x):
    ...:         print("Pavlos's function. All rights reserved")
    ...:         return original_function(x)
    ...:     return packaging

In [21]: def square(x):
    ...:     return x**2

In [27]: square(5)
Out[27]: 25


In [22]: processed_square = decorator_function(square)

In [23]: processed_square(5)
Pavlos's function. All rights reserved
Out[23]: 25
```

Function call includes the copyright message





Digestion Time

Function Decorators

```
In [19]: def decorator_function(original_function):
    ...:     def packaging(x):
    ...:         print("Pavlos's function. All rights reserved")
    ...:         return original_function(x)
    ...:     return packaging
```

Without decorator

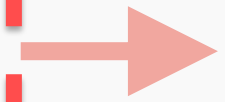
```
square = lambda x:x**2
```

```
square_sum = decorator_function(square)
```

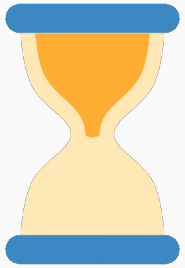
The function you declared below the decorator gets written over. Now it no longer takes only one input

Using decorator

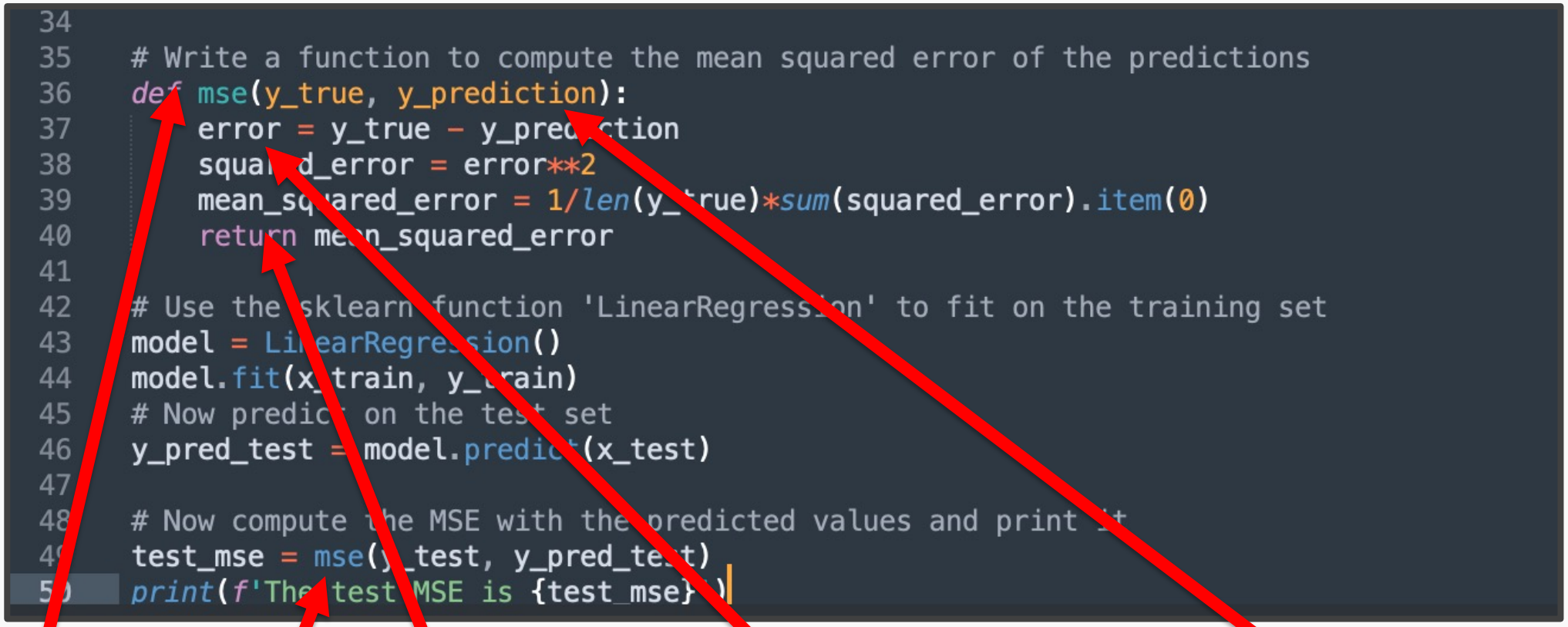
```
@decorator_function
def square(x):
    return x**2
```



```
square = decorator_function(square)
```



Digestion Time


34
35 # Write a function to compute the mean squared error of the predictions
36 def mse(y_true, y_prediction):
37 error = y_true - y_prediction
38 squared_error = error**2
39 mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
40 return mean_squared_error
41
42 # Use the sklearn function 'LinearRegression' to fit on the training set
43 model = LinearRegression()
44 model.fit(x_train, y_train)
45 # Now predict on the test set
46 y_pred_test = model.predict(x_test)
47
48 # Now compute the MSE with the predicted values and print it
49 test_mse = mse(y_test, y_pred_test)
50 print(f'The test MSE is {test_mse}')
what is def
What is mse(...)
what is return
what is local variable
What are arguments

What is pandas ?

What is with ?

What is open ?

What is a csv ?

What is len() ?

What is list comprehension ?

~~what is def~~

~~what is return~~

~~What is mse(…)~~

What is import ?

What is as ?

What is the * operator ?

What is a conditional list comprehension ?

What is range() ?

~~What are local variables~~

```
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']
x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')

# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```