When you run the file hello\_world.py, the ending .py indicates that the file is a Python program. Your editor then runs the file through the Python interpreter, which reads through the program and determines what each word in the program means. For example, when the interpreter sees the word print, it prints to the screen whatever is inside the parentheses.

When an error occurs in your program, the Python interpreter does its best to help you figure out where the problem is. The interpreter provides a traceback when a program cannot run successfully. A traceback is a record of where the interpreter ran into trouble when trying to execute your code.

A name error usually means we either forgot to set a variable’s value before using it, or we made a spelling mistake when entering the variable’s name.

Input () is used to get values from the console

name = "tinshu "

print(name.rstrip().\_\_len\_\_())

print(name.\_\_len\_\_())

# to remove it permanently , I think immutability comes here

name= name.rstrip();

print(name.\_\_len\_\_())

But be aware that you can sometimes get an arbitrary number of decimal

places in your answer:

>>> **0.2 + 0.1**

0.30000000000000004

>>> **3 \* 0.1**

0.30000000000000004

This happens in all languages and is of little concern. Python tries to find a way to represent the result as precisely as possible, which is sometimes difficult given how computers have to represent numbers internally.

***Avoiding Type Errors with the str() Function***

Often, you’ll want to use a variable’s value within a message. For example, say you want to wish someone a happy birthday. You might write code like this:

*birthday.py* age = 23

message = "Happy " + age + "rd Birthday!"

print(message)

You might expect this code to print the simple birthday greeting, Happy 23rd birthday! But if you run this code, you’ll see that it generates an error:

Traceback (most recent call last):

File "birthday.py", line 2, in <module>

message = "Happy " + age + "rd Birthday!"

u TypeError: Can't convert 'int' object to str implicitly

This is a *type error*. It means Python can’t recognize the kind of information you’re using. In this example Python sees at u that you’re using a variable that has an integer value (int), but it’s not sure how to interpret that value. Python knows that the variable could represent either the numerical value 23 or the characters *2* and *3*. When you use integers within strings like this, you need to specify explicitly that you want Python to use the integer as a string of characters. You can do this by wrapping the variable in the str() function, which tells Python to represent non-string values as strings:

age = 23

message = "Happy " + str(age) + "rd Birthday!"

print(message)

age = 20;  
msg = 'happy ' + str(age) + ' Birthday'  
print(msg)

***Integers in Python 2***

Python 2 returns a slightly different result when you divide two integers:

>>> **python2.7**

>>> **3 / 2**

1

Instead of 1.5, Python returns 1. Division of integers in Python 2 results in an integer with the remainder truncated. Note that the result is not a rounded integer; the remainder is simply omitted. To avoid this behavior in Python 2, make sure that at least one of the numbers is a float. By doing so, the result will be a float as well:

>>> **3 / 2**

1

>>> **3.0 / 2**

1.5

>>> **3 / 2.0**

1.5

>>> **3.0 / 2.0**

1.5

data = 3/2  
print(data)\_

# ouput is 1.5

What Is a List?

A list is a collection of items in a particular order

firstList = [1, 2, 3, 4, 5, 6]  
print(firstList)  
print(firstList[0])  
# gives you the last element in the array  
print(firstList[-1])

Python has a special syntax for accessing the last element in a list. By asking for the item at index -1, Python always returns the last item in the list:

This code returns the value 'specialized'. This syntax is quite useful, because you’ll often want to access the last items in a list without knowing exactly how long the list is. This convention extends to other negative index values as well. The index -2 returns the second item from the end of the list, the index -3 returns the third item from the end, and so forth.

**Changing, Adding, and Removing Elements**

Most lists you create will be dynamic, meaning you’ll build a list and

then add and remove elements from it as your program runs its course

Inserting Elements into a List

You can add a new element at any position in your list by using the insert() method

**Removing an Item Using the del Statement**

If you know the position of the item you want to remove from a list, you can use the del statement

print(firstList)  
firstList.append(300)  
print(firstList)  
firstList.insert(1,500)  
print(firstList)  
del firstList[1]  
print(firstList)

**Removing an Item Using the pop() Method**

Sometimes you’ll want to use the value of an item after you remove it from a list

ele = firstList.pop()  
print(ele)  
print(firstList)

The pop() method removes the last item in a list, but it lets you work

with that item after removing it

**Popping Items from any Position in a List**

You can actually use pop() to remove an item in a list at any position by

including the index of the item you want to remove in parentheses

ele1 = firstList.pop(0)  
print(ele1)  
print(firstList)