Understanding the difference between Statements and Expressions

- A Statement is an instruction that the Python interpreter can execute. We have only seen the assignment statement so far. Some other kinds of statements that we'll see shortly are while statements, for statements, if statements, and import statements. (There are other kinds too!)
- An Expression is a combination of values, variables, operators, and calls to functions. Expressions need to be evaluated. If you ask Python to print an expression, the interpreter evaluates the expression and displays the result.

```
# Assignment statement
x = 3
```

```
print(1 + 1)
print(len("Hello Team AlmaBetter"))
```

```
2
21
```

- In this example `len()` is a built-in Python function that returns the number of characters in a string. We've previously seen the `print()` and the `type()` functions, so this is an example of another function!
- The evaluation of an expression produces a value, which is why expressions can appear on the right hand side of assignment statements. A value all by itself is a simple expression, and so is a variable. Evaluating a variable gives the value that the variable refers to.

```
y = 3.14
```

```
y
```

```
3.14
```

```
a = 'Mighty'
# b = a == True
```

```
a == True
```

```
False
```

- Note that when we enter the assignment statement, `y = 3.14` , only the prompt is returned. There is no value. This is due to the fact that statements, such as the assignment statement, do not return a value. They are simply executed.

```
# Import statement
import pandas as pd
```

```
print(y)
```

```
3.14
```

```
y
```

```
3.14
```

- On the other hand, the result of executing the assignment statement is the creation of a reference from a variable, y, to a value, 3.14. When we execute the `print()` function working on y, we see the value that y is referring to. In fact, evaluating y by itself results in the same response.

- Instructions that a Python interpreter can execute are called statements. For example, `a = 1` is an assignment statement. `if` statement, `for` statement, `while` statement, etc. are other kinds of statements which will be discussed later.

Multi-line Statements

- In Python, the end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character ().

```
x = 1
y = 2
+ 5
z = 3
```

```
y
```

```
2
```

```
a = 1 + 2 + 3 + 4
```

```
a
```

```
10
```

- This is an explicit line continuation. In Python, line continuation is implied inside parentheses `( )`, brackets `[ ]`, and braces `{ }`. For instance, we can implement the above multi-line statement as:

```
my_list = [1]
my_tuple = (1,)
```

```
my_list, my_tuple
```

```
([1], (1,))
```

```
a = (1 + 2 + 3 + 4 +5 + 6 + 7 + 8 + 9 + 10,)
```

```
a
```

```
(55,)
```

```
type(a)
```

- Here, the surrounding parentheses `( )` do the line continuation implicitly. Same is the case with `[ ]` and `{ }`. For example:

```
a = 1
b = 2
```

```
colors = 'Set theory is the mathematical theory of well-determined collections, called sets, of objects that are called members, or elements
```

```
colors = 'Set theory is the mathematical theory of well-determined collections, called sets,\
of objects that are called members, or elements, of the set. Pure set theory deals exclusively \
with sets, so the only sets under consideration are those whose members are also sets.\
The theory of the hereditarily-finite sets, namely those finite sets whose elements are also finite sets,\
 the elements of which are also finite, and so on, is formally equivalent to arithmetic. \
 So, the essence of set theory is the study of infinite sets, and \
 therefore it can be defined as the mathematical theory of the actual—as opposed to potential—infinite.'
```

```
colors
```

```
'Set theory is the mathematical theory of well-determined collections, called sets,of objects that are called members, or elements, of
the set. Pure set theory deals exclusively with sets, so the only sets under consideration are those whose members are also sets.The th
eory of the hereditarily-finite sets, namely those finite sets whose elements are also finite sets, the elements of which are also fini
te, and so on, is formally equivalent to arithmetic.  So, the essence of set theory is the study of infinite sets, and  therefore it ca
```

- We can also put multiple statements in a single line using semicolons, as follows:

```
a = 1
b = 2
c = 3
```

```
c
```

```
3
```

```
a = 1
b = 2
c = 3
```

```
a, b, c = [1, 'a', True], (45, 67), {'Name' : 'Vikash' , 'Age' : 27}
```

```
a
```

```
[1, 'a', True]
```

```
b
```

```
(45, 67)
```

```
c
```

```
{'Name': 'Vikash', 'Age': 27}
```

```
a, b = b, a
```

```
a
```

```
(45, 67)
```

```
b
```

```
[1, 'a', True]
```

Python Comments

- Comments are very important while writing a program. They describe what is going on inside a program, so that a person looking at the source code does not have a hard time figuring it out.
- You might forget the key details of the program you just wrote in a month's time. So taking the time to explain these concepts in the form of comments is always fruitful.
- In Python, we use the hash (#) symbol to start writing a comment.
- It extends up to the newline character. Comments are for programmers to better understand a program. Python Interpreter ignores comments.

```
# This is a comment
# print('Hello')
print('Hello')
```

- We can have comments that extend up to multiple lines. One way is to use the hash(#) symbol at the beginning of each line. For example:

```
# This is a long comment
# and it extends
# to multiple lines
```

- Another way of doing this is to use triple quotes, either ''' or """.
- These triple quotes are generally used for multi-line strings. But they can be used as a multi-line comment as well. Unless they are not docstrings, they do not generate any extra code.

```
'''This is also a
perfect example of
ahjcdjh
sdjbdkjas
mnbscjsab
bkjabfa
multi-line comments'''

a = 3
print(a)
```

```
3
```

Indentation in Python

It is important to keep a good understanding of how indentation works in Python to maintain the structure and order of your code. We will touch on this topic again when we start building out functions!

- Let's create a simple statement that says: "If a is greater than b, assign 2 to a and 4 to b"
- Take a look at how to write a simple if statement

```
a=2
b=1

# if b > a:
#   a = 5
#   b = 5
```

```
a
```

```
2
```

```
b
```

```
1
```

```
a, b = 1, 2
```

```
if a < b :
  print('a is less than b')
```

```
a is less than b
```

- You'll notice that Python is less cluttered and much more readable than other languages like C or Java. How does Python manage this?
- The statement is ended with a colon, and whitespace(tab) is used (indentation) to describe what takes place in case of the statement.
- Another major difference is the lack of semicolons in Python.
- Semicolons are used to denote statement endings in many other languages, but in Python, **the end of a line is the same as the end of a statement.**
- Lastly, to end this brief overview of differences, let's take a closer look at indentation syntax in Python vs other languages:

Conditional Statements

- `if` Statements in Python allows us to tell the computer to perform alternative actions based on a certain set of results.
- Verbally, we can imagine we are telling the computer:"Hey if this case happens, perform some action"
- We can then expand the idea further with `elif` and `else` statements, which allow us to tell the computer: "Hey if this case happens, perform some action. Else, if another case happens, perform some other action. Else, if *none* of the above cases happened, perform this action."
- Let's go ahead and look at the syntax format for `if` statements to get a better idea of this:

if case1:

```
perform action1
```

elif case2:

```
perform action2
```

else:

```
perform action3
```

```
x=20   # Statement 1 - Assignment statement

if x == 30 :
  print('x is greater than 10!')
  print(f'x is {x}')
```

- `if` statement evaluates the True condition

```
x == 20
```

```
True
```

```
if False:
  print('It was true!')
```

```
if False:
  print('It was true!')
```

Let's add in some else logic:

```
x = False
y = True

if x:
  print('y was True!')
else:
  print('I will be printed in any case where x is not true')
```

```
I will be printed in any case where x is not true
```

- Let's get a fuller picture of how far `if`, `elif`, and `else` can take us!
- We write this out in a nested structure. Take note of how the `if`, `elif`, and `else` line up in the code. This can help you see what `if` is related to what `elif` or `else` statements.

```
city = 'Pune'

if city == 'New Delhi':
  print('Welcome to the Indian Capital')
elif city == 'Mumbai':
  print('Welcome to the Financial Capital of India!')
else:
  print('Where are you?')
```

```
Where are you?
```

- Note how the nested `if` statements are each checked until a True boolean causes the nested code below it to run. You should also note that you can put in as many `elif` statements as you want before you close off with an `else`.
- Let's create two more simple examples for the `if`, `elif`, and `else` statements:

```
person = 'Johnny'

if person == 'John':
    print(f'Welcome {person}!')
else:
    print("Welcome, what's your name?")
```

```
Welcome, what's your name?
```

```
number1 = 2
number2 = 9

if number1 > 0:
  if number2 < 10 :
    print(number2 + number1)
  else:
    print("Should be greater than 10")

elif number1 < 0:
    print("Ah, that's a negative number")
else:
    print("The number is zero!")
```

```
11
```

Truthy & Falsy

A "truthy" value will satisfy the check performed by if or while statements. We use "truthy" and "falsy" to differentiate from the boolean values True and False.

All values are considered "truthy" except for the following, which are "falsy":

- `None`
- `False`
- `0`
- `0.0`
- `[]` - an empty `list`
- `{}` - an empty `dict`
- `()` - an empty `tuple`
- `''` - an empty `str`

```
# Consider the following statement

if False:
  print("What do you say?")
else:
  print("I have nothing to say")
```

```
I have nothing to say
```

```
if [None]:
  print("Well I had a really bad day!")
else:
  print("Come on you aced your test.")
```

```
Well I had a really bad day!
```

```
if {1:2,3:4}:
  print("This is True")
else:
  print("This is False")
```

```
This is True
```

Start coding or generate with AI.

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.