#interprator :- Interpreter. An interpreter **translates code into machine code, instruction by instruction** - the CPU executes each instruction before the interpreter moves on to translate the next instruction. Interpreted code will show an error as soon as it hits a problem, so it is easier to debug than compiled code.

Remember that syntax is the rules of how code is constructed, while semantics are the overall effect the code has. It is possible to have syntactically correct code that runs successfully, but doesn't do what we want it to do.

Functions: - Pieces of code that perform unit of work

Keywords : - Reserved words that are used to construct instructions

“” : - string (both python and cpp)

‘’ : - string python but char cpp

\*\* -> power

**First Programming Concepts Cheat Sheet**

**Functions and Keywords**

Functions and keywords are the building blocks of a language’s syntax.

Functions are pieces of code that perform a unit of work. In the examples we've seen so far, we've only encountered the print() function, which prints a message to the screen. We'll learn about a lot of other functions in later lessons but, if you're too curious to wait until then, you can discover all the functions available [here](https://docs.python.org/3/library/functions.html).

Keywords are reserved words that are used to construct instructions. We briefly encountered for and in in our first Python example, and we'll use a bunch of other keywords as we go through the course. For reference, these are all the reserved keywords:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False | class | finally | is | return |
| None | continue | for | lambda | try |
| True | def | from | nonlocal | while |
| and | del | global | not | with |
| as | elif | if | or | yield |
| assert | else | import | pass |  |
| break | except | in | raise |  |

You don't need to learn this list; we'll dive into each keyword as we encounter them. In the meantime, you can see examples of keyword usage [here](https://www.programiz.com/python-programming/keyword-list).

**Arithmetic operators**

Python can operate with numbers using the usual mathematical operators, and some special operators, too. These are all of them (we'll explore the last two in later videos).

* **a + b** = Adds a and b
* **a - b** = Subtracts b from a
* **a \* b** = Multiplies a and b
* **a / b** = Divides a by b
* **a \*\* b** = Elevates a to the power of b. For non integer values of b, this becomes a root (i.e. a\*\*(1/2) is the square root of a)
* **a // b** = The integer part of the integer division of a by b
* **a % b** = The remainder part of the integer division of a by b

**Datatypes : -**

1. integer: -
2. float: -
3. string: -
4. bool: - False True
5. complex: - using complex(rp,ip) function where rp-> real part and ip-> imaginary part both float
6. ……..

type() to tell which class(datatype it belong to)

When attempting to mix incompatible data types, you may encounter a **TypeError**. You can always check the data type of something using the type() function.

Variables: -Names that we give to certain values in our programs

Expression: - A combination of numbers, symbols, or other variables that produce a result when evaluated

As we saw earlier in the video, some data types can be mixed and matched due to implicit conversion. Implicit conversion is where the interpreter helps us out and automatically converts one data type into another, without having to explicitly tell it to do so.

By contrast, explicit conversion is where we manually convert from one data type to another by calling the relevant function for the data type we want to convert to. We used this in our video example when we wanted to print a number alongside some text. Before we could do that, we needed to call the *str()* function to convert the number into a string. Once the number was explicitly converted to a string, we could join it with the rest of our textual string and print the result.

**Functions**

def function\_name(par\_1,par\_=default):

body

return return\_value

**Floor division // giver integer part of division**

You nailed it! In Python uppercase letters are alphabetically sorted before lowercase letters

Building off of the if and else blocks, which allow us to branch our code depending on the evaluation of one statement, the elif statement allows us even more comparisons to perform more complex branching. Very similar to the if statements, an elif statement starts with the elif keyword, followed by a comparison to be evaluated. This is followed by a colon, and then the code block on the next line, indented to the right. An elif statement must follow an if statement, and will only be evaluated if the if statement was evaluated as false. You can include multiple elif statements to build complex branching in your code to do all kinds of powerful things!

**Loops**

While for recursion

Important : - right initializing

range(n) -> 0 1 2 3 ……. n-1

for can be used to interate over any variable….

for i in range(1,10,2): ## only interger

range(a,b,c) a-> start default = 0 b-> end ( till b-1 is calculated) c-> size of each jump

The range() function can take a third parameter, too. This third parameter lets you  alter the size of each step. So instead of creating a sequence of numbers incremented by 1, you can generate a sequence of numbers that are incremented by 5.

To quickly recap the range() function when passing one, two, or three parameters:

* One parameter will create a sequence, one-by-one, from zero to one less than the parameter.
* Two parameters will create a sequence, one-by-one, from the first parameter to one less than the second parameter.
* Three parameters will create a sequence starting with the first parameter and stopping before the second parameter, but this time increasing each step by the third parameter.

print(“ eefq ”, end = “”) end speicifies the end of print default is \n.

## Loops Cheat Sheet

## Loops Cheat Sheet

Check out below for a run down of the syntax for while loops and for loops.

### While Loops

A while loop executes the body of the loop while the condition remains True.

Syntax:

1

2

while condition:

    body





Things to watch out for!

* **Failure to initialize variables.** Make sure all the variables used in the loop’s condition  are initialized before the loop.
* **Unintended infinite loops.** Make sure that the body of the loop modifies the variables used in the condition, so that the loop will eventually end **for all possible values of the variables.**

Typical use:

While loops are mostly used when there’s an unknown number of operations to be performed, and a condition needs to be checked at each iteration.

### For Loops

A for loop iterates over a sequence of elements, executing the body of the loop for each element in the sequence.

Syntax:

1

2

for variable in sequence

    body





The range() function:

range() generates a sequence of integer numbers. It can take one, two, or three parameters:

* range(n): 0, 1, 2, ... n-1
* range(x,y): x, x+1, x+2, ... y-1
* range(p,q,r): p, p+r, p+2r, p+3r, ... q-1 (if it's a valid increment)

**Common pitfalls:**

* **Forgetting that the upper limit of a range() isn’t included.**
* **Iterating over non-sequences.** Integer numbers aren’t iterable. Strings are iterable letter by letter, but that might not be what you want.

Typical use:

For loops are mostly used when there's a pre-defined sequence or range of numbers to iterate.

### Break & Continue

You can interrupt both while and for loops using the break keyword. We normally do this to interrupt a cycle due to a separate condition.

You can use the continue keyword to skip the current iteration and continue with the next one. This is typically used to jump ahead when some of the elements of the sequence aren’t relevant.

If you want to learn more, check out this [wiki page on for loops](https://wiki.python.org/moin/ForLoop).