🎄 What is Python?

Python is an interpreted, high-level, general-purpose programming language. In short, Python is highly available on many computers already and is very easy to write.

Python is available in all Linux operating systems, the TryHackMe Attack Box, Elven Toy Making Machinations, and Mac OS (but sadly, not Windows by default).

🎅 Installing & Tooling

The TryHackMe AttackBox already has Python installed. To use it, load up a terminal and then type "python3". This will load an interactive editor for Python.

We'll be creating files so we won't need to use the interactive editor. Instead, to run a file with Python execute: "python3 hello.py". We'll create hello.py next.
But if you do not have Python installed, you'll need to install it. Follow this link and install Python for your system. Make sure to install a text editor such as VS Code which will help you write Python code.

🤶 Hello, World!

It is traditional for your first program to be "Hello, World!" so let's do it.

Open a new file in VS Code (or your text editor of choice) and call it *hello.py*. The .py extension means it is a Python file. Now type:

```
print("Hello, World!")
```

Save the file, and then in a Terminal navigate to where the file is and run *python3 hello.py* You should see the output on your screen as "Hello, World!".

Let's break down what each of these components means. Let's explain what this does:

👼 print()

Print is a function. We give it some text and it'll print it to the screen. We'll see functions a lot in Python. They are essential, as they allow us to re-use code.

To define our own function, we use the def keyword.

```
def hello():
    print("Hello, World!")
```

We use a tab (or 4 spaces) inside functions to denote that the code belongs to that function. This is called a scope, and unfortunately, we won't go into it too much here. But I'll link some resources at the end which will.

👰 "Hello, World!"

Is what we call a string (a string of characters). It's just text.

👰 Variables

Now in the last section, I said "String (a string of characters)".

What does that mean? In programming, we need to have data types. Every bit of data has a type in common with it. You already know some.

If I said: 1, 2, 3, 4, 5, 6, 7, 8, 9 "Are these sentences?" No! They're numbers. See, you already know data types 😜

In Python, it's the same. We have some essential data types that hold things:

- String (a string of characters)
- Integer - a whole number (-50, 50, 60, 91)
- Float - a floating-point number (21.3, -5.1921)
- List - a list of items ([1, 2, 3], ["hi", 6, 7.91])

And more....

```
hello = "Hello, World!"
```

We use the equals sign as an assignment operator. It assigns the value on the right-hand side to the bucket on the left.

Now let's say we wanted to add this variable to another variable. A common misconception is that we take the bucket itself and use that. But in Python, we don't. We **pass by reference**. As in, we merely pass a location of the variable — we do not pass the variable itself. The alternative is to pass by value. This is very important to understand, as it can cause a significant amount of headaches later on.

This is very important in toy making. We once had a small bug where an elf assigned different variables to the same toy. We thought we had 800 versions of the toy as we had 800 variables, but it turns out they were all pointing to the same toy! Luckily those children managed to get toys that year.

 Operators

Let's talk about operators. An operator is something between 2 variables/values and does something to them. For example, the addition operator:

```
x = 3 + 1 # x = 4
```

Python supports many maths operators:

```
3 + 1
```

```
3 / 1 # divided by
```

```
3 * 4 # times

2 ** 2 # 2 to the power of 2

2 % 3 # 2``mod 3
```

Now the cool thing, operators don't just work on numbers. They work on strings too. And lists. And dictionaries.

```
[3, 2] + [6, 7] # [3, 2, 6, 7]

"Hello, " + "World!" # "Hello, World!"
```

🍪 Boolean

The two values for the data type boolean are True and False. Much like Santa's list of Naughty and Nice, it is either True or False (never both).

True and False are extremely valuable. In binary, 1 represents True and 0 represents False. Through these 2 values, we can represent all data on a computer, provided we are using logic gates. Those logic gates appear in Python as operators.

We'll go through one you may already know:

```
True or False # True
```

The or operator returns true when either the left side or right side is True.

Let's quickly go through all the others

```
True and True # True
```

This returns True if and only if both the left and right sides are True

```
not True # False
```

not negates the right-hand side expression. So the opposite of True is False.

We can negate the Or statement like so:

```
not (True or False) # False
```

Now it only returns True when both sides of the or are False.

🥛 If Statements

If statements are one of the most powerful statements in all programming. We want to do something if a condition is met. If a child has been nice, they get a toy. Else they get coal!

So for example, if the list is not empty:

```
x = [6] if x: print("I run!")
```

Because x is non-empty, it has a truthy value. Which means this reads as "If True" and it runs. If it read as "If False" it does not run.

If statements only run if the condition is True.

Now let's see it in action.

```
name = input("What is your name? ")
```

We can take user input using the input() function. Now let's say we are a wide bouncer at Santa's Grotto. We only want to let people in that have a name in our special club. First, we need to create a list of names. Can you guess what we'll be using for this?

```
names = ["Skidy", "DorkStar", "Ashu", "Elf"]
```

Great! And now if we have a string, such as Jabba, how do we check if the name is in the list?

```
"Jabba" in names # False
```

Now we want to print a special little message if and only if the users inputted name appears in our list. We can do this with:

names = ["Skidy", "DorkStar", "Ashu", "Elf"]

```
name = input("What is your name? ")

if name in names:
    print("The Wide One has allowed you to come in.")
```

Now, what if their name does not appear in the list of names? We can use an else clause.

```
names = ["Skidy", "DorkStar", "Ashu", "Elf"]

name = input("What is your name? ")

if name in names:

    print("The Wise One has allowed you to come in.")

else:

    print("The Wise One has not allowed you to come in.")
```

🎁 One Line If Statements

We can also build one line if statements pretty nicely in Python.

```
age = 12 name = "Jabba" if age == 12 else "Skidy"
print(name) # 'Jabba'
```

❄️ Loops

Loops allow us to perform the same code, repeatedly. For loops are best described how they are read:

```
names = ["Skidy", "DorkStar", "Ashu", "Elf"]

for name in names:
    print(name)
```

For every name in the list of names, do something (in this case — print the name).

For loops can iterate over the elements of any iterable. Let's look at a function which returns an iterable, range.

Range returns a list of numbers in a range. So to loop between 1 and 9 we would do:

```
range(1, 9)
```

Range is inclusive, so 1 and 9 are included. Now to loop over this: `for i in range(1, 9): print(i)` Note: We often use `i` as the variable in a for loop as it stands for "item".

Libraries

You've seen how to write code yourself, but what if we wanted to use other peoples code? This is called *using a library* where a *library* means a bunch of someone else's code. We can install libraries on the command line using the command: `pip install X` Where *X* is the library we wish to install. This installs the library from PyPi which is a database of libraries. Let's install 2 popular libraries that we'll need:

- Requests
- Beautiful Soup

```
pip3 install requests beautifulsoup4
```

Something very cool you can do with these 2 libraries is the ability to extract all links on a webpage.

```
# Import the libraries we downloaded earlier
# if you try importing without installing them, this step will fail
from bs4 import BeautifulSoup
import requests

# replace testurl.com with the url you want to use.
# requests.get downloads the webpage and stores it as a variable
html = requests.get('testurl.com')

# this parses the webpage into something that beautifulsoup can read over
soup = BeautifulSoup(html, "lxml")
# lxml is just the parser for reading the html

# this is the line that grabs all the links # stores all the links in the links variable
links = soup.find_all('a href')
for link in links:
    # prints each link
    print(link)
```

This was a very short introduction to Python, but here are some more links if you wanted to learn more:

- Python Zero to Hero
- Python Moduluo Operator in Practice
- Automate the Boring Stuff with Python