PYTHON Reference (from web)

A computer program is a set of instructions that allow us to do a task automatically on a computer. We can make a computer program in a programming language. Computer programs are generally called "software". With a computer program we can do all sorts of things. Some examples are calculators, video games, text processors, browsers, and all the things you have ever used in a computer. Nowadays, there are computers everywhere. Any device such as a cell phone, smart watch, or modern car is running software that was made in programming. To begin, we are going to learn python, which is one of the easiest programming languages to learn.

Let’s begin writing python! We are not going to explain each detail of python independently. For that, you could read the python documentation, which is located here:

<https://docs.python.org/3.9/tutorial/index.html>

However, if you don’t know any programming, going directly to the documentation can be overwhelming. We are just going to explain some parts of python which are a good start to begin to write your own programs to exploit software. We do this by making examples that achieve one objective and we explain how they work along the way. This will allow you to read code written by someone else, of course, with the help of google if they use elements that you did not know previously.

When you are learning a programming language, there is a tradition in which the first program you write simply prints "Hello World!"" on the screen. We will be using python 3, the number 3 is the version of python. Let’s start doing the "hello world!" program.

Open the shell server, and create a folder called "python\_examples". You can do it with the following line:

mkdir python\_examples

Now, access that folder using

cd python\_examples

Create a file called "helloworld.py", you can do it with:

nano helloworld.py

To make our 'hello world!' program in python requires just one line of code! Simply write this on the file:

print("Hello World!")

Now save the file in nano by pressing 'control' and 'x' at the same time, and then press 'y', then 'enter'.

Run the program on the terminal with:

python3 helloworld.py

You should see that "Hello World!" is printed on the screen when you run it:

python3 helloworld.py

Hello World!

That was our first program in python!

Python, as any other programming language, has variables. A variable can hold different types of data. What we just printed on the screen was a string of characters. When we enclose something in quotes, we are telling python it is a string of characters. A string is a data type. In python, to create a variable we simply choose a name and assign the value that we want. For example, we are going to create a variable called my\_string, and we are going to assign to that variable the value "Hello World!":

my\_string = "Hello World!"

That line of code makes the variable my\_string equal to "Hello World!". In python programming, the symbol = is used to assign the value from the right side of the equal to the variable at the left side. Variables can have any name we like, except some specific words that are reserved for python instructions. For example, the word 'print' is reserved, so you cannot use it as a variable name.

Now, if we print the variable, it should print "Hello World!". Do that experiment next. The python script should look like this:

my\_string = "Hello World!"

print(my\_string)

Run it and you will see "Hello World!"" printed on the screen again.

Hello World!

You can also assign numbers to variables and do mathematical operations between them. Let’s make a simple program that calculates the area of a square. Create a file called "area.py" and write the following:

side1 = 4

side2 = 8

result = side1 \* side2

print(result)

If you run that script, what do you think is going to print?

When you run it you should see:

32

Those were very trivial examples. Now, suppose you want to print a list of 20 numbers that starts at 0 and ends at 19. We can do that in just a couple of lines, instead of writing 20 prints! Create a file called loop.py and use the following code:

**for** i **in** range(20):

print(i)

Run it and you should see:

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

We have introduced the concept of a python loop. The word 'for' is used to declare a 'for loop', which is a loop that iterates in a range of numbers. The 'i' next to 'for', is a variable that will be incremented on each iteration on a range of 20. We can change the range for a bigger one or a smaller one by changing the number inside the parenthesis. Note that a line of code will be inside the loop, if it is indented by four spaces. For example, run this:

**for** i **in** range(10):

print("I am inside the loop")

print(i)

print("I am OUTSIDE")

You will see:

I am inside the loop

0

I am inside the loop

1

I am inside the loop

2

I am inside the loop

3

I am inside the loop

4

I am inside the loop

5

I am inside the loop

6

I am inside the loop

7

I am inside the loop

8

I am inside the loop

9

I am OUTSIDE

Note that the string "I am OUTSIDE" was printed only once, because it is outside the loop. To be inside the loop the code needs to be indented by 4 spaces, as we said. Once we use a line of code that is not indented for the first time after the loop, that is considered the end of the loop. If you try to indent a line after the loop has finished, like this:

**for** i **in** range(20):

print("I am inside the loop")

print(i)

print("I am outside")

print("I am outside 2")

That would cause a syntax error when you run it. A syntax error means that the code is not complying with the way python should be written. In this case, would specifically show an indentation error:

python3 helloworld.py

File "helloworld.py", line 5

print("I am outside 2")

^

IndentationError: unexpected indent

That happens because we put and indentation, and the for loop was already closed. Syntax errors at the beginning can happen to you by accident and you might not fix them very easily, but with a little time you will begin to fix them quickly if they happen. To practice, spot the syntax error in the following code:

**for** i **in** range(20):

prin("I am inside the loop")

print(i)

print("I am outside")

What is the error?

Run it to see what happens. It will show:

python3 helloworld.py

File "helloworld.py", line 2

prin("I am inside the loop")

^

SyntaxError: invalid syntax

Python shows you the line with the error, but not the exact location. In this case we missed the 't' from 'print'. Another error might be that the colon from the for loop is missing:

**for** i **in** range(20)

print("I am inside the loop")

print(i)

print("I am outside")

In that case it will show you:

python3 helloworld.py

File "helloworld.py", line 1

for i in range(20)

^

SyntaxError: invalid syntax

If you add the missing colon after range(20), the program should work. A syntax error can happen because any reserved word is misspelled; remember that reserved words are words that python recognize as instructions. For example, 'print', 'for', 'in' are reserved words in our program. Additionally, a syntax error can happen because of a missing symbol such as a colon.

As a challenge, implement a program that prints your name 10 times, and below your name prints a number starting at 100 and ends at 109. The output of your program should look similar to:

Samuel

100

Samuel

101

Samuel

102

Samuel

103

Samuel

104

Samuel

105

Samuel

106

Samuel

107

Samuel

108

Samuel

109

Hint: use range(100, 110).

Once you are done with the previous challenge, fix the following program that has several syntax errors and make it work:

**for** i inn range(10:

prnt(i)

The program should print the numbers from 0 to 9.

So far, we have seen how a computer can repeat an instruction several times, which is something fundamental in a computer. We want computers to do repetitive tasks for us. Another fundamental functionality we want in computers is conditional clauses. A conditional clause means that a program will do an action only if a condition is met or take another path if the condition is not met. For example, suppose you are printing the numbers from 0 to 9, and you want to print a message when the number is less than 5 and another message when the number is equal or greater than 5. You would do it in the following manner:

**for** i **in** range(10):

**if** i < 5:

print("The following number is less than 5")

**if** i >= 5:

print("The following number is greater than or equal to 5")

print(i)

Run it and verify the results. We have introduced an if-clause, which is a conditional clause. Note that all the code is inside the loop. The first message is inside the first if-clause, that is only fulfilled when 'i' is less than 5. The second message is inside the second if-clause, which is only fulfilled when the 'i' is greater than or equal to 5. At last, we print the variable 'i', which is not inside any if-clause, so it is always printed.

Another way to implement this program, is using an 'else':

**for** i **in** range(10):

**if** i < 5:

print("The following number is less than 5")

**else**:

print("The following number is greater than or equal to 5")

print(i)

When then condition in an if-clause is not met, it enters the 'else' to execute what is inside. You should still see this output when you run the program:

python3 helloworld.py

The following number **is** less than 5

0

The following number **is** less than 5

1

The following number **is** less than 5

2

The following number **is** less than 5

3

The following number **is** less than 5

4

The following number **is** greater than **or** equal to 5

5

The following number **is** greater than **or** equal to 5

6

The following number **is** greater than **or** equal to 5

7

The following number **is** greater than **or** equal to 5

8

The following number **is** greater than **or** equal to 5

9

To practice, implement a program that prints a range of 100 numbers and prints a different message when the numbers are smaller than 10, another message when the numbers are between 10 and 50, and another message when the numbers are greater than 50.

3.1. Lists

There are several data structures in python, which are simply structures to organize data in a certain manner. Different data structures have different properties. We are going to introduce one that is called a 'list', which allows us to store several values, one after the other.

We create a list like this:

my\_list = ["I", "Love", "picoCTF"]

print(my\_list)

We can iterate in the list to operate on each item in any way we want. For example, suppose we want to print each item of the list, we could do this:

my\_list = ["I", "Love", "picoCTF"]

print(len(my\_list))

print(my\_list)

**for** i **in** my\_list:

print(i)

When you run that program, you should see the following output:

3

['I', 'Love', 'picoCTF']

I

Love

picoCTF

Note that the number 3 printed is the length of the list. You can sort the list alphabetically by calling a function that is part of the list like this:

my\_list = ["this", "is", "not", "ordered", "alphabetically"]

my\_ordered\_list = my\_list.sort()

**for** i **in** my\_list:

print(i)

You should see this output when you run that program:

alphabetically

**is**

**not**

ordered

this

Now, create a list of numbers, and print it backwards! Using google, it should be very easy to find how to do it.

3.2. Functions

If you have a piece of code that you want to use often, copy pasting that piece of code is a bad idea because your code gets longer and for a human becomes harder to read. On the other hand, if you want to make a modification in that piece of code, you will have to modify every part in which you copy and pasted that code. We can overcome that by using functions. A function can receive parameters, which are variables you pass to the function so operations with them can be done. Additionally, a function can return a value, which is the result after all the operations are done. Let’s see an example of a function that verifies if a number is even or odd. If it is even, it will return True. If it is odd, it will return False. The program receives any number you input and verifies such an input. Note that the '%' operator in the code is the modulo operator, which calculates the remainder. In this case we calculate the remainder of x divided by 2 and compare that to zero to determine if the number is even or odd. Read the code to understand!

**def** even\_odd(x):

**if** (x % 2 == 0):

**return** **True**

**else**:

**return** **False**

print("Input a number:")

my\_number = input()

**if** even\_odd(my\_number):

print("The number is even")

**else**:

print("The number is odd")

Run that program and try several numbers!

3.3. Input and output

A program might need to have interactions with a user. For example, a calculator expects that the user enters some numbers to then do the processing. Receiving user input in a terminal is very easy in python because it has predefined functions that do it for us. The function ‘input()’ waits until the user writes something in the terminal and presses enter. Note that a function can have zero parameters. Then, the function returns the string that the user wrote, and we assign it to the variable number\_iterations’. Here is an example, in which we allow the user to control the number of iterations of our program:

print("Input the number of iterations:")

number\_iterations = input()

**for** i **in** range(number\_iterations):

**if** i < 5:

print("The following number is less than 5")

**else**:

print("The following number is greater than or equal to 5")

print(i)

Run that program. When you run it, it will do nothing until you input a number in the terminal and press enter.

In other cases, the data we want to input does not have to come from the user. It could come from a file. We can read all the lines from a file using the function 'open'. Create a file called “pico.txt” in the same folder that you are creating the python programs. Then, in that file copy and paste this text:

The Cosmos **is** all that **is** **or** was **or** ever will be.

Our feeblest contemplations of the Cosmos stir us

-- there **is** a tingling **in** the spine,

a catch **in** the voice,

a faint sensation,

**as** **if** a distant memory,

of falling **from** **a** great height.

We know we are approaching the greatest of mysteries.

Save the file. Now, in the same folder, create a program with the following code:

filepath = "pico.txt"

i = 1

**with** open(filepath, "r") **as** my\_file:

**for** line **in** my\_file:

print(i)

print(line)

i += 1

You should see the following output when you run the program:

1

The Cosmos **is** all that **is** **or** was **or** ever will be.

2

Our feeblest contemplations of the Cosmos stir us

3

-- there **is** a tingling **in** the spine,

4

a catch **in** the voice,

5

a faint sensation,

6

**as** **if** a distant memory,

7

of falling **from** **a** great height.

8

We know we are approaching the greatest of mysteries.

As you saw, this program reads a file and enumerates each line in the output. The 'open' function has two parameters, the first one is the path of the file you want to open, and the second has a string with the letter 'r', which means that we want to **r**ead the file. 'my\_file' is just the name of the file we want to read. Then, we can iterate over each of the lines of the file in a for loop.

Note that this is all made inside a 'with' block. We use the 'with' statement before opening a file to close the file automatically after reading. Also, to handle possible exceptions during the execution. What that means is that when you open a file, you must close it and make sure that it closes correctly. For example, if you do my\_file.close(), that would close the file. Imagine that along the way before calling close, something happens and you never get to the line in which you close the file, so you left it open accidentally. Later we will give you more details on exceptions. For the time being, just think of 'with' as an easy way to ensure that the file will be closed correctly.

If you want to save your output in another file, you can easily do it in the following manner:

filepath\_read = "pico.txt"

filepath\_write = "outputpico.txt"

i = 1

**with** open(filepath\_read, "r") **as** file\_read:

**with** open(filepath\_write, "w") **as** file\_write:

**for** line **in** file\_read:

file\_write.write(str(i) + "**\n**")

file\_write.write(line + "**\n**")

i += 1

print("look inside your folder...")

We introduced some new concepts in this code. This:

str(i)

Is a cast from an integer to string. We want to convert that integer into a string to be able to concatenate two strings. For example, if we have the string "hello" and the integer 123, and we want to create a string that is "hello123", we can concatenate those two values. But first, we need to convert the integer to string, otherwise python will show an error. To concatenate strings, we use the operator '+'. When we add two strings, python will concatenate them. When we add two integers, python will do a mathematical addition. To represent a break of line in a string, we use "\n".

After this explanation, you should know that this:

str(i) + "**\n**"

Simply converts an integer to string, and then we concatenate a break line to it. We do that, because the function line write() does not add a breakline to the string after it writes it, so we would have a file with a single huge line of text if we don’t do that. When you run the code, you should see no output in the terminal, but if you show the contents of the folder you are in, you should see a new file called 'outputpico.txt'. If you show the contents of that file, you should see the following:

cat outputpico.txt

1

The Cosmos is all that is or was or ever will be.

2

Our feeblest contemplations of the Cosmos stir us

3

-- there is a tingling in the spine,

4

a catch in the voice,

5

a faint sensation,

6

as if a distant memory,

7

of falling from a great height.

8

We know we are approaching the greatest of mysteries.

We just learned how to read and create files!

3.4. Comments

It is a good practice to explain what your code is doing in a comment. In that way, the reader of the code, which may be yourself, will understand what some part of the code is doing. You will realize that when you write some code, you will forget the exact logic and you will have to read it again to understand what you did. In summary, comments are something very important in programming. In python, you write a comment by adding the '#' symbol at the beginning of any line of your code. This line, will be ignored by the python interpreter as it did not exist, so it does nothing in the program. See the following example:

print("Input the number of iterations")

*# We read user input and assign it to the variable number\_iterations*

number\_iterations = input()

*# We iterate according to the value input by the user*

**for** i **in** range(number\_iterations):

**if** i < 5:

*# We only print this message when the value of i is less than 5*

print("The following number is less than 5")

**else**:

*# We only print the value of i is greater than or equal to 5*

print("The following number is greater than or equal to 5")

*# We always print this*

print(i)

3.5. Try-except and exceptions

Exceptions are useful in hacking in several cases, for example, when you want an attack to keep executing even if an unknown error occurred. When a program tries to execute an instruction that even though it has a correct syntax, it cannot be done for some other reason, an exception is thrown. For example, if you try to divide a number by zero, that can have the correct syntax to do it, but when the program is executing the line it will stop and fail. Let’s do the experiment:

num1 = 8

print("Input the number that will divide:")

num2 = input()

result = num1 / num2

print(result)

print("The program keeps executing to do other stuff...")

As you can see the program divides 8 by any number input by the user. If you run it and input for example 2, nothing bad will happen, and you will see this:

Input the number that will divide:

2

4

The program keeps executing to do other stuff...

Now, run the program again and input 0, you will see this:

Input the number that will divide:

0

Traceback (most recent call last):

File "helloworld.py", line 4, **in** <module>

result = num1 / num2

**ZeroDivisionError**: integer division **or** modulo by zero

An error ocurred because you cannot divide by zero. That is a rule of python and most programming languages. Your program will stop when an error happens, further lines will not be executed. In this case, you could verify that the number is not zero in an if-clause. For this example, let’s fix the program instead using a try-except:

num1 = 8

print("Input the number that will divide:")

num2 = input()

**try**:

result = num1 / num2

print(result)

**except**:

print "An error has occurred, did you try to divide by zero?")

print("The program keeps executing to do other stuff...")

In our previous code, you would print the same message for any error. Try to input a string instead of 0. It will show the same message. If you want to be more specific, you can catch specific errors in the following manner:

num1 = 8

print("Input the number that will divide:")

num2 = input()

**try**:

result = num1 / num2

print(result)

**except** **ZeroDivisionError**:

print("Do not divide by zero, that is forbidden.")

**except** **TypeError**:

print("Your input value must be an integer.")

print("The program keeps executing to do other stuff...")

Now when you input a string, it will show this:

Input the number that will divide:

"Any string"

Your input value must be an integer.

The program keeps executing to do other stuff...

And if you input zero it will show this:

Input the number that will divide:

0

Do **not** divide by zero, that **is** forbidden.

The program keeps executing to do other stuff...

Note that when an error occurs, the following lines inside the 'try' block will not execute. See that 'result' is not printed, and that makes sense because there was no result to print. The program jumps into the ‘except’ block immediately.

3.6. Pass arguments to a python program

When you call a program from the command line, it is possible to pass arguments in the same way you do with several programs in the terminal. The following program shows how to do this:

**import** **sys**

print('Number of arguments:', len(sys.argv), 'arguments.')

print('Argument List:', str(sys.argv))

print("The number of arguments is taken from the second argument.")

print("Remember that in an array [0] is the first one, [1] is the second one.")

number\_iterations = sys.argv[1]

f = open("output2.txt", "w")

**for** i **in** range(int(number\_iterations)):

**if** i < 5:

f.write("The following number is less than 5")

**else**:

f.write("The following number is greater than or equal to 5")

f.write(str(i))

f.close()

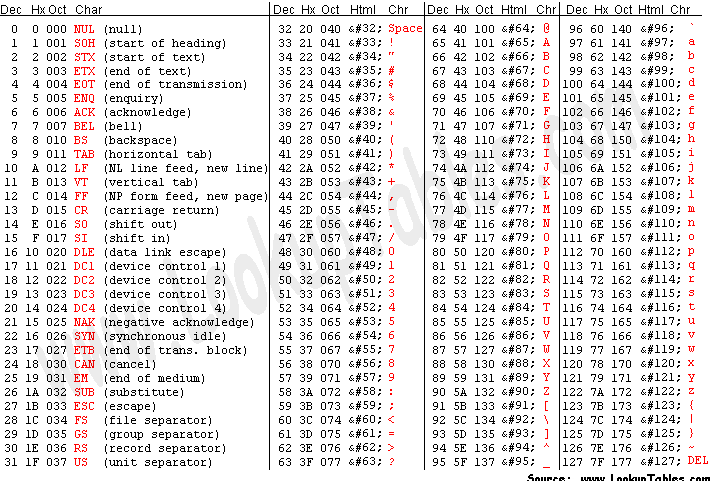
print("look inside your folder...")

In this example we introduced another way to open a file, which does not handle the close for us.

3.7. ASCII

ASCII is a way in which a computer represents characters. We could say that in memory a computer only stores numbers, but a program can interpret those numbers in a certain way to understand them as characters.

In the following table, it is shown what number corresponds to each character in ASCII:



*Figure 2. ASCII Table Ref*[*http://www.asciitable.com*](http://www.asciitable.com/)

The ASCII includes all the characters that are used in the English language. For other languages, there is a bigger character set called Unicode.

For example, in the ASCII table, you can see that the @ symbol is represented as the 64 number in decimal.

The table also has a column called Hx or Hexadecimal, which is base 16. Decimal is base 10.

The decimal base is the one we use in everyday life, which likely comes from the fact that humans have 10 fingers. Therefore, we have 10 different symbols to represent all different numbers. In computers, it is helpful to have a base with 16 symbols because it translates easier to binary. You probably know that most computers physically store only binary numbers, which are represented only by 0 and 1. A **b**inary dig**it** is called a bit. Although computers use binary, base 16 is easy to translate from binary for us humans.

The hexadecimal base (or base 16) has the following symbols:

0 1 2 3 4 5 6 7 8 9 a b c d e f

The binary base (or base 2) has these symbols:

0 1

The decimal base (or base 10), has the following symbols:

0 1 2 3 4 5 6 7 8 9

Let’s see in python how can we use the hexadecimal representation to print characters. In a python string, you can put “\x” which is a special sequence to tell python that the following two characters are a hexadecimal number:

print("picoCTF")

print("**\x70\x69\x63\x6f\x43\x54\x46**")

When you run that program you should see:

picoCTF

picoCTF

Check the table to see that the characters match!

As a challenge, print the string “I\_LOVE\_PICOCTF” only using hexadecimal. Note that uppercase letters are represented by a different hexadecimal number than lowercase letters.

3.8. Pwn tools

For binary exploitation, there is a very useful library called pwn tools:

<http://docs.pwntools.com/en/stable>

Keep this library in mind as an important part of python for exploitation. You do not need to learn anything right now. We will teach how to use it in binary exploitation.

3.9. Http requests in python

Below is an example of how you can request a web page in python. Here we are requesting the HTML of the picoCTF website. Right now, maybe you do not know HTML and worry this will not make much sense to you. After you are done with the Web section, come back here and try this example:

**import** **http.client**

conn = http.client.HTTPSConnection("https://picoctf.com/")

conn.request("GET", "/")

r1 = conn.getresponse()

print(r1.status, r1.reason)

*# 200 OK*

data1 = r1.read()

conn.request("GET", "/")

r2 = conn.getresponse()

print(r2.status, r2.reason)

*# 404 Not Found*

data2 = r2.read()

conn.close()