CODE

CODE: Make Your Own Fun

Python makes a great language to have on Z

12 steps 120 minutes

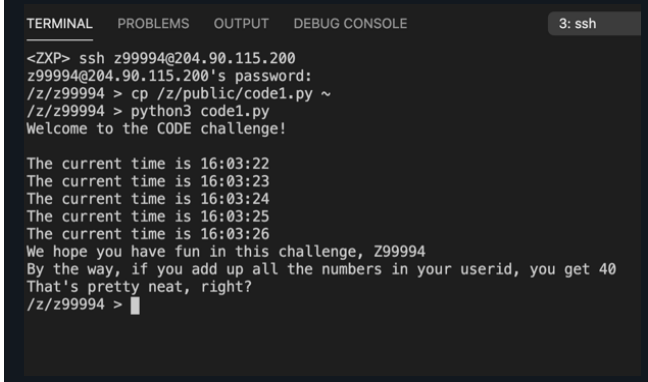
THE CHALLENGE

You'll be introduced to the concept of coding here. Or, if you're already familiar with variables, loops, and if statements, you'll be quickly getting a refresher and learning that it doesn't matter what platform you're on, Python code looks like Python code.

We'll be using the Python programming language, as it's perhaps the most widely used language right now, and it makes it very easy to get started with simple code that won't overwhelm you if this is your first-time coding.

BEFORE YOU BEGIN

All you need to get started is access to a USS shell, either through an SSH program, a terminal window, or really any other method. All challenges here take place on the z/OS system, which you access with your zxxxxx id.



1. IF YOU WANT TO CODE THEN...

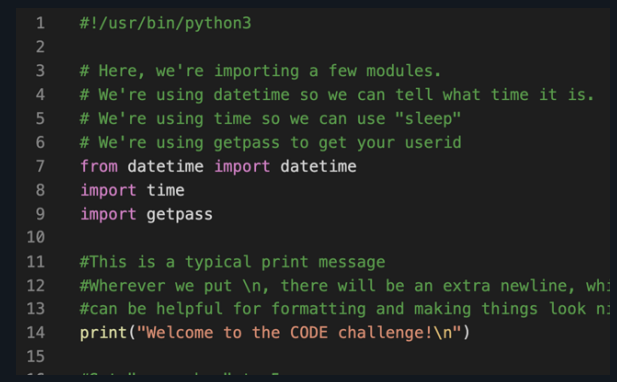
SSH into the system and if you don't have a code1.py file, copy the code1.py program from /z/public, either by copy and pasting through VS Code, or entering the command

cp /z/public/code1.py ~

(as shown in the above screenshot)

Next, run the program with ***python3 code1.py***

This says "Use the Python interpreter to run the code1.py program in this directory" and look at the output. As you can see, there's a lot of things happening in here, but in the next step, we'll look under the hood and see what's going on.

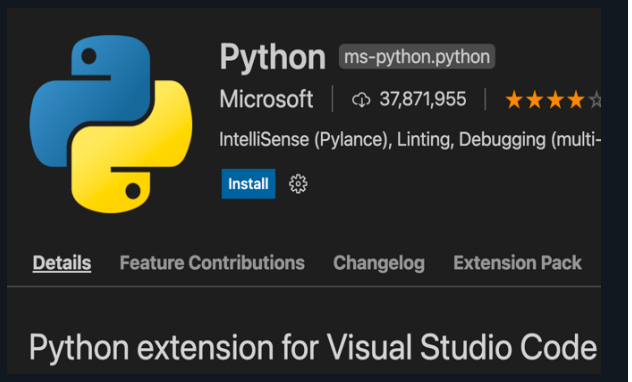


2. LET'S SEE WHAT'S INSIDE

Open up the code in an editor window within VS Code. There's code to look at in here, and lots of comments (those are the lines in green starting with #)

Even if you have never programmed before, you can see the part where it counts backwards from 5, where it reads your userid, where it figures out what time it is, and how it stops for a second between each section. You may find some of this information helpful as you tackle later steps. *(that's a hint right there)*

You don't need to do anything more with this, just know it's here.



3. A WORD ON EXTENSIONS

Extensions in VS Code are a nice way to gain additional functions, but they can also make things not work right.

You might get a message asking if you want to install the Python extension. If you have a Python extension installed, you can try to keep using it, but you will probably get warning messages in later Python-related challenges. If you want your editor to look like the instructions here, don't install any extensions besides the ones outlined in the instructions.

All good? Let's jump to Step #4.

```
1  #!/usr/bin/python3
2
3  #Here is where we are setting our var
4  #Remember in algebra class how you mig
5  #It's just like that.
6  the_letter = "z"
7  the_word = "pumpkin"
8
9  #This could have been written as   if
10 # by using variables, it makes it so
```

4. A NAME BY ANY OTHER NAME

Copy over **code2.py** from /z/public and look at the code in there. This program takes a word and figures out if the letter "Z" is in it.

Notice how we set "*the_word*" to "pumpkin" on line 7. We're creating what's known as a *variable*. A variable is a placeholder for a value in a program, and it can be letters, numbers, or really any sort of data. In this case, we're using it to represent the word we want to investigate.

Run the program and see what happens.

```
1  #!/usr/bin/python3
2
3  #Here is where we are setting our vari
4  #Remember in algebra class how you migh
5  #It's just like that.
6  the_letter = "z"
7  the_word = "pizza"
8
9  #This could have been written as   if
0  # by using variables, it makes it so w
1  # the actual code below when we want t
```

5. KIND OF ANTICLIMACTIC...

Let's fix that.

Edit the code so instead of "pumpkin" the variable *word* gets set to "pizza", and run the program again. You should see a message confirming that pizza does in fact have the letter Z in it.

Feel free to experiment with other words, and letters, and then look at the part of the code that does the checking. Make sure to save your file (going up to File -> Save, or using the keyboard shortcut) to commit any changes you make between edits and running the code.

```
8
9  #This could have been written as
10 # by using variables, it makes it
11 # the actual code below when we wa
12 # letters and words.
13 if the_letter in the_word:
14     print("Yes! The letter " + the_le
15 #else:
16 # print("Nope. The letter " + the_
```

6. BLOCKS FOR THE BETTER

A big part of writing and understanding code is figuring out the *logic* of a program. In other words, the way it flows. Code can generally be organized into *blocks* of code that get run in order, or if a condition is met.

This program uses an *if* statement on line 13. It's testing to see if *the_letter* is in *the_word*.

An *if* statement has the condition being tested, followed by a colon (:) and then the code that gets run if that condition is met underneath it.

We also indent that code so we know that anything shifted to the right of that *if* statement underneath it is what's going to happen if the condition is met.

In some languages, indentation is not mandatory, it's just a nice thing to have. Python requires indentation.

"WHY DO WE NEED MORE THAN ONE LANGUAGE? CAN'T I JUST LEARN ONE?

Different languages are better at different things. COBOL is a language built around high-precision and low-latency transactions. C and C++ are good for system utilities and high-performance applications that benefit from lots of control over memory and other system resources.

Python is an extensible language that's been around for a long time and used for a lot of things. You can call it a General-Purpose Programming Language, as you can use it for AI (Artificial Intelligence), mathematical simulations, web applications, games, automation... just about anything.

If you're going to learn one language, it makes sense for it to be Python. From there, once you have the fundamentals down, it's just a matter of learning each language's particular points. Code on!

```
7   the_word = "pumpkin"
8
9   #This could have been written as
10  # by using variables, it makes it s
11  # the actual code below when we war
12  # letters and words.
13  if the_letter in the_word:
14      print("Yes! The letter " + the_let
15  else:
16      print("Nope. The letter " + the_le
```

7. OR ELSE!!!!

Lines 15 and 16 add another step to the *if* statement, an *else* clause. So if the condition being tested on line 13 is not met (there is no z in the word) it will say something else.

Right now, these lines of code are commented out, so they don't actually do anything. Delete the hash marks (#), save the file, and then run it again. Make note of the indentation, the *else* should line up with the *if*, and the next line should be indented to show that it belongs to the *else* above it.

Test it out and make sure it works for z and non-z words.

```
#!/usr/bin/python3

marbles = 10  #You start out with 10 marbles
marble_dots = "**********" #Pretend these are ten marb

while (marbles > 0):

    #This line of code prints out the "marble_dots" varial
    # but will only include the number of characters in i
    # for how many marbles are left.
    print(marble_dots[:marbles])

    #This prints out how many marbles you have left.
    # We have to say str(marbles) because marbles is a nu
```

8. DON'T LOSE YOUR MARBLES

Copy the **marbles.py** program from /z/public into your home directory and give it a try. Hopefully it's pretty straightforward. Open up the file in your editor to check out the code.

An important thing to notice is how the message about how many marbles are left appears in the code only once but gets printed out ten times.

This is because we have used a *while loop* starting at line 6. A *while loop* says, "keep doing this as long as the following thing is true". In this case, as long as there are more than 0 marbles left.

```
You have 5 marbles left.

You have 4 marbles left.

You have 3 marbles left.
Warning: You are running low on marbles!!

You have 2 marbles left.
Warning: You are running low on marbles!!

You have 1 marbles left.
Warning: You are running low on marbles!!
```

9. THE FINAL COUNTDOWN

Once you've traced your way through the code, take a look at lines 13-14 That looks like a special feature that you can enable by uncommenting. Erase the comment marks so the code is active and run the program again.

The code was *supposed* to issue a warning message when there are three or fewer marbles left, but something is wrong with the logic in that *if* statement. OH NO!!!

See if you can fix it so the message prints out just like the screenshot above, for when there's 3 or fewer marbles left.

A few hints to get you started:
>= means "greater than or equal to"
< means "less than"
Consider where this *if* statement is in relation to the part of the code that subtracts one marble. Hmmm...

There are many different ways you can fix this code.

"HOW ELSE CAN I CONTROL CODE?"

We're really just skimming the surface of things in this challenge, enough to give someone with no programming experience a chance to check out what coding looks like with lots of examples.

If you're into this and want to learn more, be sure to read more about what you can do with Python at IBM Developer. There are videos, tutorials, podcasts, code patterns, and lots of projects you can get started on, no matter what your skill level is.

Even if you don't plan to be a fulltime coder, it's important to have some familiarity with code so you can fix problems when they come up, and maybe even add features to code someone else wrote.

```
*******
You have 7 marbles left.

******
You have 6 marbles left.

*****
You have 5 marbles left.

****
You have 4 marbles left.

***
You have 3 marbles left.
Warning: You are running low on marbles!!
```

10. I'M A VISUAL LEARNER

Numbers are nice, but sometimes it's nice to visualize things.

Your next task is to make the program print out the number of asterisks. So when there's 5 marbles left, you get *****

How are we going to get this done? One of the variables we created, *marble_dots*, is just ten asterisks in a row. Maybe you were curious what we're doing with that.

The following line of code will print out six marble dots.
print(marble_dots[:6])

If you replace the 6 with a variable, it will print out whatever the variable is set to. Are the wheels turning yet?

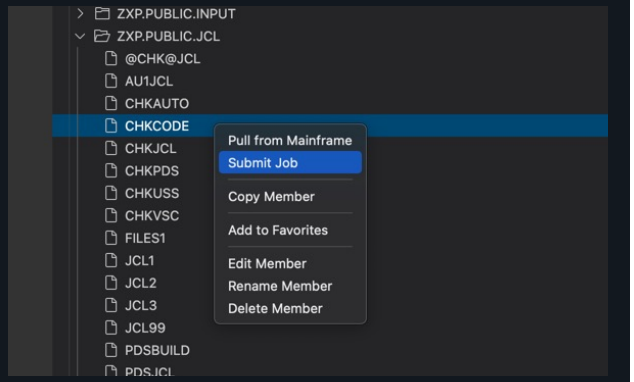
Figure out where you want to put that line of code and how to make it work so that the result looks like the output in the screenshots above.

```
***
You have 3 marbles left.
Warning: You are running low on marbles!!

**
You have 2 marbles left.
Warning: You are running low on marbles!!

*
You have 1 marbles left.
Warning: You are running low on marbles!!

You are all out of marbles
```

11. ALL OUT OF MARBLES

We want the user to know when the program is done, and that there are no more marbles left.

Make the message say "You are all out of marbles" and make sure it appears last, and only once in your program.



12. DOUBLE-CHECK, SUBMIT

Now it's time to run the CHKCODE job and see if you crossed all of your T's and dotted all of your lowercase I's. Make sure you have the right number of "marbles", that the spacing looks right, and that your "all out of marbles" message looks just like it does in the screenshot.

We'll be checking your marbles.py code, so make sure it's all in that file, and that it works as described.

This is just the beginning. We'll do even more with Python in the next challenge, so hopefully you've enjoyed this.

NICE JOB! LET'S RECAP

In most things, the hardest part is getting started. If this was your first time hacking away at code, it might have felt difficult, frustrating, and even a little weird. We picked an example that would illustrate a few of the most important concepts: variables, loops, if/then/else statements, modifying variables, and using library functions.

Completing this challenge means you're able to envision a problem, mentally solve it, and then implement the solution in code. That's a huge talent, and we suggest taking a small break of your own before continuing to the next challenge. But only if you want.

NEXT UP...

Wrapping up the Fundamentals