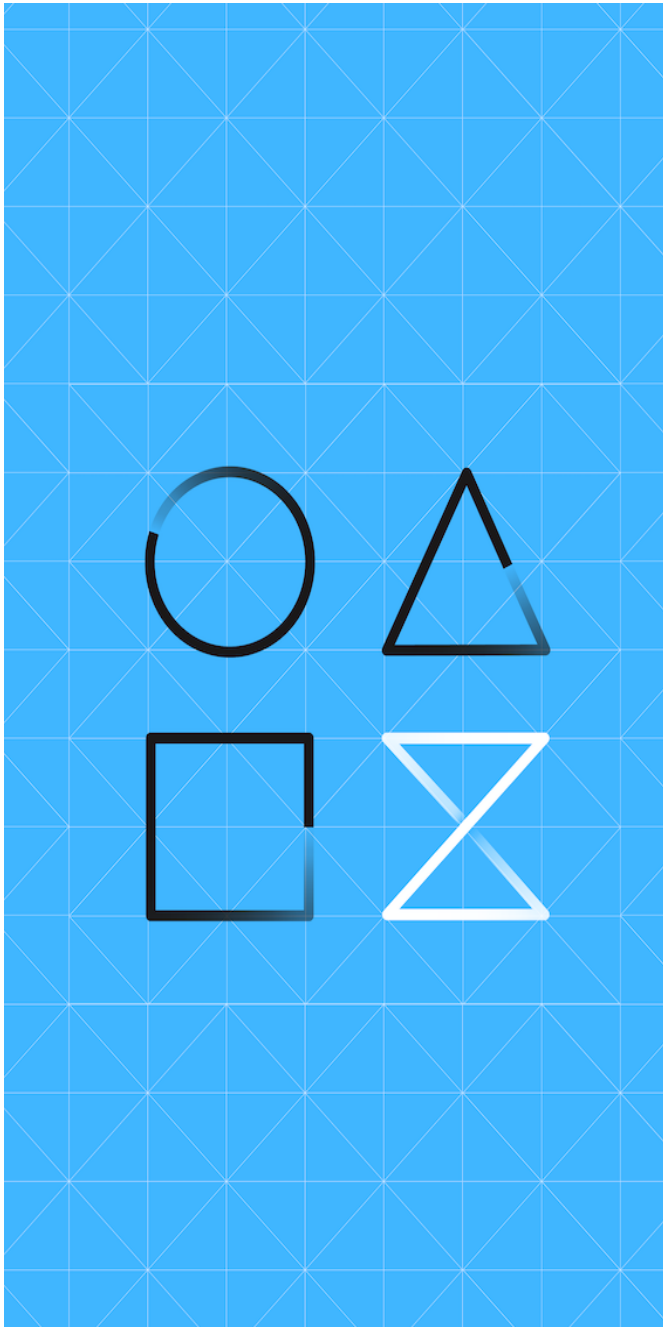


CODE1

Make Your Own Fun

- [PYTHON IS A GREAT LANGUAGE TO HAVE ON Z](#)
- [1 IF YOU WANT TO CODE, THEN ...](#)
- [2 LET'S SEE WHAT'S INSIDE](#)
- [3 A WORD ON EXTENSIONS](#)
- [4 A ROSE BY ANY OTHER NAME](#)
- [5 A BIT OF AN ANTICLIMAX ?](#)
- [6 BLOCKS FOR THE BETTER](#)
- [7 OR ELSE!!](#)
- [8 DON'T LOSE YOUR MARBLES](#)
- [9 THE FINAL COUNTDOWN](#)
- [10 I'M A VISUAL LEARNER](#)
- [11 ALL OUT OF MARBLES](#)
- [12 DOUBLE-CHECK; SUBMIT](#)



PYTHON IS A GREAT LANGUAGE TO HAVE ON Z

The Challenge

You'll be introduced to the basic concepts 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.

You'll be using the `Python` programming language, as it's perhaps the most widely-used language for new development 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

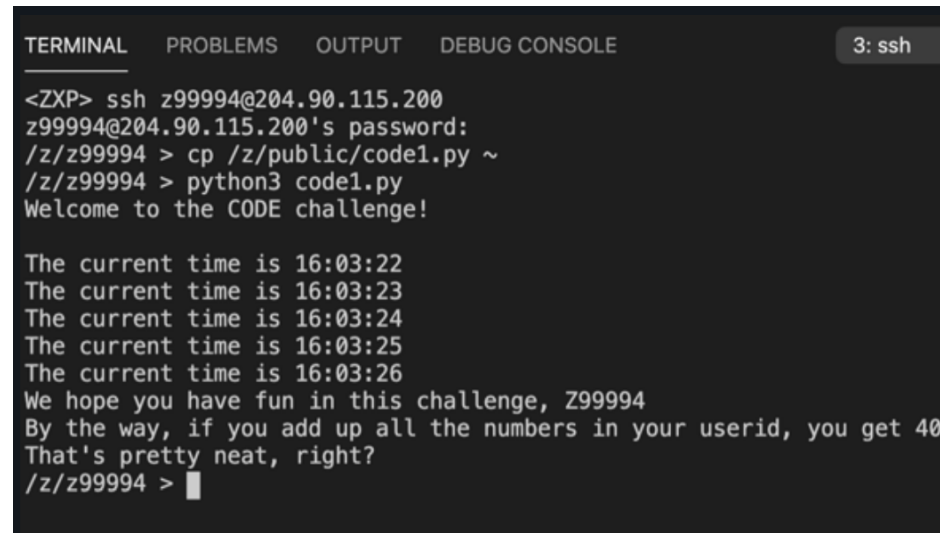
All you need to get started is access to the `USS` shell, either through an `SSH` program, or a terminal window.

The only technical challenge you need to have completed before this one is `VSC1`.

Investment

Steps	Duration
12	120 minutes

1 IF YOU WANT TO CODE, THEN ...



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  3: ssh
<ZXP> ssh z99994@204.90.115.200
z99994@204.90.115.200's password:
/z/z99994 > cp /z/public/code1.py ~
/z/z99994 > python3 code1.py
Welcome to the CODE challenge!

The current time is 16:03:22
The current time is 16:03:23
The current time is 16:03:24
The current time is 16:03:25
The current time is 16:03:26
We hope you have fun in this challenge, Z99994
By the way, if you add up all the numbers in your userid, you get 40
That's pretty neat, right?
/z/z99994 > █
```

Log into the IBM **z/OS** system using **ssh**

Check your home directory for a file called “code1.py” (using the **ls** command); if you do not have one, copy the original **code1.py** program from **/z/public**, either by copy-and-pasting through **VSCode**, or by entering the following 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 are lot of things happening in here; in the next step, you will take a look “under the hood” and see what’s going on inside.

CODE1240630-2221

2 LET'S SEE WHAT'S INSIDE

Open up the code in an editor window within `VSCode`.

There is a small amount of code to look at in here, and lots of comments (those are the lines in green starting with #)

```
1  #!/usr/bin/python3
2
3  # Here, we're importing a few modules.
4  # We're using datetime so we can tell what time it is.
5  # We're using time so we can use "sleep"
6  # We're using getpass to get your userid
7  from datetime import datetime
8  import time
9  import getpass
10
11 #This is a typical print message
12 #Wherever we put \n, there will be an extra newline, wh
13 #can be helpful for formatting and making things look r
14 print("Welcome to the CODE challenge!\n")
15
```

Even if you have never programmed before, you can see the part where it

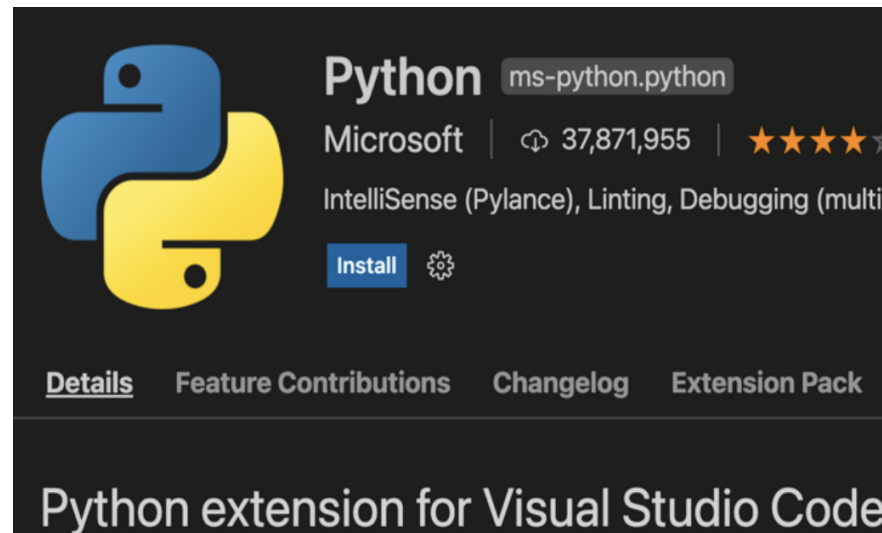
- counts backwards from 5
- reads your userid
- figures out what time it is
- and how it stops for a second between each section.

You may find some of this information helpful as you tackle later steps. **(that's a hint, right there!)**

3 A WORD ON EXTENSIONS

Extensions in `VSCode` are a nice way to gain additional functions, but they can also make things not work as expected.

You might get a message from `VSCode` asking if you want to install the Python extension.



If you do have a `Python` extension installed, you can try to keep using it, but be aware you will probably get warning messages in later `Python`-related challenges.

One to look out for in particular is an error message telling you that the 'zoautil_py' module cannot be found – this is *NOT* a problem for you working on these challenges.

If you want your editor to look like the screenshots here, don't install any extensions besides the ones outlined in the challenges.

All good? Let's move on ...

4 A ROSE BY ANY OTHER NAME

Make sure you have a copy of over code2.py in your home directory.

```
1  #!/usr/bin/python3
2
3  #Here is where we are setting our var
4  #Remember in algebra class how you mi
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
```

This Python program takes a word and figures out if the letter “z” is in it.

Notice how ‘the_word’ to set to “pumpkin” on line 7.

This is 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, it is used to represent the word you want to “investigate” - to check if it contains a “z”.

The program also has a variable for holding the value of the particular letter you use for the investigation.

Read the program to understand what it should do; then run the program and see what happens (use `python3 code2.py`).

5 A BIT OF AN ANTICLIMAX ?

Nothing happens, right?

Let's fix that!

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

Edit the code so instead of “pumpkin” the variable word is set to “pizza”, save the file, and *run the program again **in your **USS** terminal session*

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.

Note: Make sure to save your file between edits (by going using the editor menu option File -> Save, or using the keyboard shortcut) to commit any changes you make, and remember to the run the code in your USS terminal session.

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.

Program code can generally be organized into blocks of code that can be run in order, or if a condition is met.

```
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_
```

This program uses an `if` statement on line 13.

It is testing to see if `'the_letter'` is in `'the_word'`.

An `if` statement has the condition being tested, followed by a colon (:) and the code which will be run, when that condition is satisfied, is in a “block” underneath it.

A block is a group of program statements that all belong together.

The block of code is “indented” (moved to the right), so you know that anything in that block is what’s going to happen if the condition is met.

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 for 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 has been around for a long time and used for a lot of different tasks. 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 just one programming language, it makes sense for it to be `Python`.

From there, once you have the fundamentals understood, it's just a matter of learning any other programming language's particular points.

Code on!

7 OR ELSE!!

```
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 wan
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
```

Lines 15 and 16 in code2.py add another step to the 'if' statement - an 'else' clause.

If the condition being tested on line 13 is not met (there is no 'z' in the word) the program can say or do something about that.

Right now, these lines of code are commented out, so they do not actually do anything.

Delete the hash marks (#) at the beginning of the lines in the 'else' block, 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 (or block or lines) 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.

8 DON'T LOSE YOUR MARBLES

In your **USS terminal session**, try to run the 'marbles.py' program from your home directory.

Hopefully, this should be pretty straightforward.

Open up the file in your editor to check out the code.

```
#!/usr/bin/python3

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

while (marbles > 0):

    #This line of code prints out the "marble_dots" variable
    # but will only include the number of characters in
    # 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 number
```

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 10 times.

This is because there is a 'while' loop starting at line 6.

A 'while' loop says, "Keep doing this set of actions as long as the following condition is true".

In this case, the condition is true as long as there are more than 0 marbles left.

9 THE FINAL COUNTDOWN

Once you have traced your way through the 'marbles.py' code, take a look at lines 13 and 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!!!

```
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!!
```

See if you can fix it so the message prints out just like the screenshot above - only when there are 3 or fewer marbles left.

A few hints to get you started:

- \geq 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?”

You have really only skimmed the surface of the [Python](#) programming 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 are keen 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 current 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.

10 I'M A VISUAL LEARNER

Numbers are nice, but sometimes it is better to visualize things.

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

How are you going to get this done?

One of the variables in the program 'marble_dots' is just ten asterisks in a row.

Maybe you were curious what that was for.

The following line of code will make the program print out six marble dots.

```
print(marble_dots[:6])
```

If you replace the '6' with a variable name, it will print out the number of asterisks for whatever the variable is set to. (Assuming the variable has been set to a number!)

Are the wheels of your mind turning yet?

```
*****
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!!
```

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 screenshot above.

CODE1240630-2221

11 ALL OUT OF MARBLES

```
***  
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
```

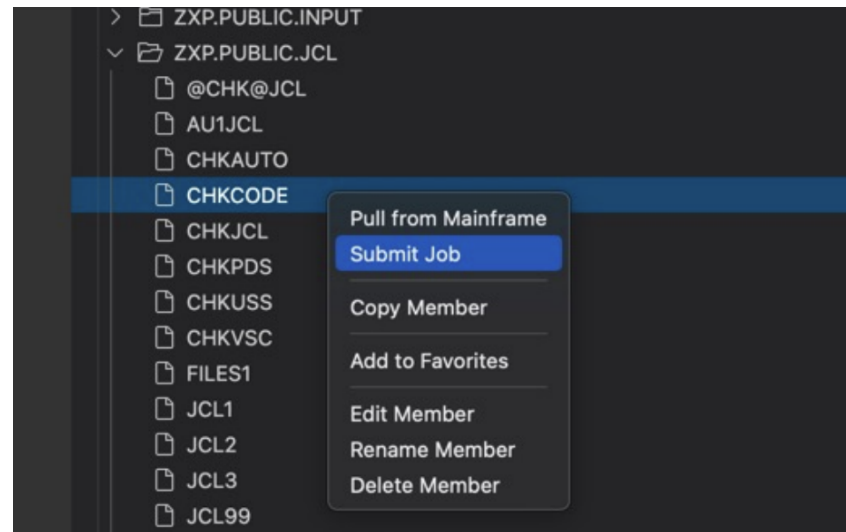
The user needs to know when the program is done, and that there are no more marbles left.

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

12 DOUBLE-CHECK; SUBMIT

Now it's time to run the `CHKCODE` job from `ZXP.PUBLIC.JCL` and see if you have a program that produces the the same output as the previous screenshot.

Make sure you have the right number of “marbles” displayed, that the spacing looks right, and that your final “all out of marbles” message looks just like it does in the screenshot.



The validation job will run your ‘marbles.py’ program, so make sure it’s all working as described.

This is just the beginning; you’ll do even more with Python in later challenges, so hopefully you’ve enjoyed this.

Nice job - let's recap	Next up ...
<p>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.</p> <p>We picked an example that would illustrate a few of the most important concepts: + variables + loops + if/then/else statements + modifying variables + using library functions.</p> <p>Completing this challenge means you're able to envision a problem, mentally solve it, and then implement the solution in code.</p>	<p>Wrapping up Fundamentals</p>