os.environ, os.getenv() and os.putenv()

The **os.environ** value is known as a **mapping** object that returns a dictionary of the user's environmental variables. You may not know this, but every time you use your computer, some environment variables are set. These can give you valuable information, such as number of processors, type of CPU, the computer name, etc. Let's see what we can find out about our machine:

```
import os
print(os.environ)

# environ({
#     "PWD": "/usercode",
#     "NODE_PATH": "/usr/local/lib/node_modules",
#     "TERM": "xterm",
#     "LEIN_VERSION": "2.7.1",
#     "HOSTNAME": "f3a04e1c7f55",
#     "PATH": "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/bin/",
#     "NODE_ENV": "development",
#     "OLDPWD": "/tmp",
#     "CLASSPATH": "/usr/local/bin/educative/java/*:.:./*",
#     "MYSQL_PWD": "educative",
#     "LEIN_ROOT": "1",
#     "LEIN_INSTALL": "/usr/local/bin/",
#     "_": "/usr/bin/python3",
#     "SHLVL": "1",
#     "HOME": "/nonexistent"
# })
```

Your output won't be the same as mine as everyone's PC configuration is a little different, but you'll see something similar. As you may have noticed, this returned a dictionary. That means you can access the environmental variables using your normal dictionary methods. Here's an example:

```
import os
print(os.environ["HOSTNAME"])

# d16e2747fa73 (It might vary every time you run it)
```

You could also use the **os.getenv** function to access this environmental variable:

```
import os
print(os.getenv("HOSTNAME"))

# edf61e3dc24d (It might vary every time you run it)
```

The benefit of using os.getenv() instead of the os.environ dictionary is that if you happen to try to access an environmental variable that doesn't exist, the getenv function will just return None. If you did the same thing with os.environ, you would receive an error. Let's give it a try so you can see what happens:

```
import os
os.environ["TMP2"]
# KeyError: 'TMP2'

print(os.getenv("TMP2"))
# None
```

Now let's try getting the same invalid environment variable using getenv (*It returns `None` instead of throwing an exception*).

```
import os

print(os.getenv("TMP2"))
# None
```

Python's `os` module also provides `os.putenv()`, which you can use to set an environment variable.

Here is an example:

```
import os

os.putenv('py_env', 'foo')
```

However this code only affects subprocesses that you start using `os.system()`, `os.popen()`, etc. If you try to get that environment variable that you just set using `os.getenv()`, it will return `None`.