## Applications of HEX

We use the hexadecimal system to store error codes during the work of various software products. For example, in some operating system errors are encoded in this way. If you decode your error code, you'll find out the exact error that occurred. Besides, in URLs, character codes are written as hexadecimal pairs prefixed with %. You can see for yourself by googling a symbol @. The link in the address bar of your browser would look like this:

**Conclusion**

As you can see, there are many ways to use HEX. In this topic, we have covered that:

* A HEX number is basically a 44-bit group with 1616 values that can include numbers from 00 to 99 and A,B,C,D,E,F*A*,*B*,*C*,*D*,*E*,*F* as hex-digits;
* Converting a HEX number into a binary requires splitting into a sequence of 44-digit binary numbers;
* In the real world, the application of HEX numbers is vast — from error codes to symbol encoding.

For instance, 63h, 0x63 and 6316​ are three different ways to indicate that 63 is a hexadecimal number.

def function\_name(parameter1, parameter2, ...):

# function's body

...

return "return value"

The names of a function and its parameters follow the same convention as **variable names**, that is, they should be written in **lowercase with underscores between words**.

## Summary

Thus, we've learned the syntax for declaring functions. Now you also know that:

* Parameters of a function are simply aliases, or placeholders for values that you will pass to them. Parameters are re-initialized every time you call the function. Inside the function, you have access to these values, which means you can perform calculations on them.
* A function can simply perform an action without returning anything or return a specific result. If your function doesn't return anything, assigning its result to a variable or printing it will give you None.
* # Function to convert miles to kilometers
* def miles\_to\_kilometers(miles):
* # Your code here
* return miles \* 1.60934
* # Main program
* if \_\_name\_\_ == "\_\_main\_\_":
* miles = float(input())
* result = miles\_to\_kilometers(miles)
* print(f"{result:.2f}")

return stops the function execution and/or sends back a value resulted from the function call

Which of the following code snippets is the best way to declare a function that determines the parity of a number?

P**arity** refers to whether a number is **even** or **odd**.

* A number has **even parity** if it is divisible by 2 (**remainder is 0** when divided by 2).
* A number has **odd parity** if it is **not** divisible by 2 (**remainder is 1** when divided by 2).

def is\_even(number):

return number % 2 == 0

A module is simply a file that contains Python statements and definitions. It usually has a **.py** extension. What really makes the module system powerful is the ability to **load** or **import** one module from another.

 **Python** = Uses **modules** (.py files with functions, classes, and variables).

 **Java** = Uses **packages** (folders) and **classes** (no modules).

 **C#** = Uses **namespaces** and **classes** (no modules).

 **JavaScript** = Has **modules**, but they work differently than Python.

import super\_module

Or

from super\_module import super\_function

A special form of import statement allows you to load all the names defined in a module. It is called [*wildcard import*](https://hyperskill.org/learn/step/6019)and has the syntax from module import \*. You should generally avoid this in your code. It can cause unexpected behavior because you don't know what names exactly are imported into the current [namespace](https://hyperskill.org/learn/step/6019). Besides, these names may shadow some of the existing ones without your knowledge. It's better to make it explicit and specify what you're importing.

In case you have to use several import statements, pay attention to their order:

1. standard library imports, i.e. from the Standard Python library
2. third party dependency imports, i.e. something installed additionally
3. local application imports, i.e. files that belong to the current application

Having your imports grouped, you may put a blank line between import sections. Also, some guidelines, including ours, recommend sorting imports alphabetically.

Python Module Index [Python Module Index — Python 3.13.2 documentation](https://docs.python.org/3/py-modindex.html)

import math  
def calculate\_area(radius):

return math.pi \* (radius\*\*2)

area = calculate\_area(5)

print(str(area))

# place `import` statement at top of the program

from math import copysign

# don't modify this code or the variables may not be available

x, y = map(float, input().split(' '))

print(copysign(x, y))

import math

# Input the number

n = int(input())

# Use the factorial function from the math module to calculate the factorial

print(math.factorial(n))

import string

print(string.digits) # Prints digits from 0 to 9

print(string.ascii\_lowercase) # Prints lowercase letters from a to z