Errors:

The first thing you need to know about programming is how to print **"Hello, world!"**. The second one is that this might be a challenging task, as even such a tiny script can contain various errors. Here you are:

In [6]:

```
print("Hello, world!"
```

```
  File "<ipython-input-6-ec501c8f224c>", line 1
    print("Hello, world!"
                         ^
SyntaxError: unexpected EOF while parsing
```

Traceback is a stack trace that appears when your code causes an error and it reports detailed information on that particular error, indicating the definite files in which the error occurred. Nonetheless, the lines that are the most informative for us right now are the last two. They point out the mistake in your code.

Common errors for beginners

Some of the most common syntax errors are:

- **wrong spelling** of keywords and function names, e.g. While instead of while, pint instead of print;
- the wrong number of **parentheses** in function calls, e.g. print "just one round bracket");
- **indents** are also the fertile soil for errors, therefore, use spaces and tabs carefully;
- **quote** Don't forget to wrap a string in quotes of the same type: triple quotes for multi-line strings, double or single quotes for ordinary strings

Exception handling:

Exception is an event, which occurs during the execution of a program that disrupts the normal flow of program instructions. When a python scripts raises an exception. It must either handle the exception immediately otherwise it terminates and quits.

The try block lets you test a block of code for errors.
The except block lets you handle the error.
The finally block lets you execute code, regardless of the result of the try and except blocks.

Example

In [4]:

```
try:
    print(x)
except:
    print("An error exception occurred")
```

```
An error exception occurred
```

Code Explain

Since We are trying to rpint the value of x variable but we did not define the x variable so it will raise errorTthe try block raise an error, the except block will be executed. Without the try block, the program will crash and raise an error.

Many exception: You can define as many as exception blocks as you want.
e.g., if you want to execute a special block of code for a special kind of error.

Example

In [19]:

```
try:
    print(x)
except NameError:
    print("Variable x is not defined")
except:
    print("Something else went wrong")
```

```
-1
```

else: You can use the else keyword to define a block of code to be executed if no errors were raised.

Example

In [3]:

```
try:
    print("Hello")
except NameError:
    print("Something went wrong")
except:
    print("Nothing went wrong")
```

```
Hello
```

finally: The finally block, if specified will be executed regardless if the try block raises an error or not.

In [4]:

```
try:
    print(x)
except NameError:
    print("Something went wrong")
except:
    print("The try except is finished")
```

```
Something went wrong
```

Raise an exception: As a python developer you can choose to throw an exception if a condition occurs.
To throw an exception use the raise keyword.

In [5]:

```
x=-1
if x<0:
    raise exception("sorry, no numbers below zero is accepted")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-447a0c321371> in <module>
      1 x=-1
      2 if x<0:
----> 3     raise exception("sorry, no numbers below zero is accepted")

NameError: name 'exception' is not defined
```

Code Explain

The raise keyword is used to raise an exception you can define what kind of errors to raise, and the test to print the user.

User-defined exceptions:

However, some programs may need user-defined exceptions for their special needs. Let's consider the following example. We need to add two integers but we do not want to work with negative integers. Python will process the addition correctly, so we can create a custom exception to raise it if any negative numbers appear. So, it is necessary to know how to work with the user-created exceptions along with the built-in ones.

1. Raising user-defined exceptions

If we want our program to stop working when something goes wrong, we can use the raise keyword for the Exception class when a condition is met. You may come across either your or built-in exceptions like the ZeroDivisionError in the example below. Note that you can specify your feedback in brackets to explain the error. It will be shown when the exception occurs.

In [9]:

```
def example_exceptions_1(x, y):
    if y == 0:
        raise ZeroDivisionError("The denominator is 0! Try again, please!")
    elif y < 0:
        raise Exception("The denominator is negative!")
    else:
        print(x / y)
```

In [11]:

```
example_exceptions_1(10, 0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-11-9f97bb626e07> in <module>
----> 1 example_exceptions_1(10, 0)

<ipython-input-9-b58314c7fbe1> in example_exceptions_1(x, y)
      1 def example_exceptions_1(x, y):
      2     if y == 0:
----> 3         raise ZeroDivisionError("The denominator is 0! Try again, please!")
      4     elif y < 0:
      5         raise Exception("The denominator is negative!")

ZeroDivisionError: The denominator is 0! Try again, please!
```

In [12]:

```
example_exceptions_1(10, -2)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-12-59c01de1acf0> in <module>
----> 1 example_exceptions_1(10, -2)

<ipython-input-9-b58314c7fbe1> in example_exceptions_1(x, y)
      3         raise ZeroDivisionError("The denominator is 0! Try again, please!")
      4     elif y < 0:
----> 5         raise Exception("The denominator is negative!")
      6     else:
      7         print(x / y)

Exception: The denominator is negative!
```

In [14]:

```
example_exceptions_1(10, 5)
```

```
2.0
```

Code Explain: If there is a zero, the program will stop working and will display the built-in exception with the message you specified; note that if we had not raised this exception ourselves, it would have been raised by Python but with a regular message. If y is a negative integer, we get the user-defined exception and the message. If the integer is positive, it prints the results.

2. Creating a user-defined exception class

we are creating a new class of exceptions named **NegativeResultError** derived from the built-in Exception class. We print the message informing that the procedure of creating the class is finished inside.

NOTE: Note that it is good to end the name of the exception class with such word as Error or Exception

In [20]:

```
class NegativeResultError(Exception):
    print("Hooray! My first exception is working!")


def example_exceptions_2(a, b):
    try:
        c = a / b
        if c < 0:
            raise NegativeResultError
        else:
            print(c)
    except NegativeResultError:
        print("There is a negative result!")
```

```
Hooray! My first exception is working!
```

In [21]:

```
example_exceptions_2(2, 5)
```

```
0.4
```

In [22]:

```
example_exceptions_2(2, -5)
```

```
There is a negative result!
```

Code Explanation: In the example_exceptions_2(a, b) function below we use the try-except block. If the result of the division is positive, we just print the result. If it is negative, we raise an exception and go to the corresponding part of the code with except to print the message.