

47 Handling exceptions

47.1 About

An exception is an unexpected condition that occurs within a program when it runs. For example, if the program tries to open a file, but the file does not exist, a “file not found” exception occurs.

Because errors like this don’t always occur we might never see them in normal usage. But we can predict what kinds of exceptions can occur and try to handle them inside the program. This is known as exception handling.

Exception handling in Python is mainly done with the **try...except...else** structure.

- **try** : run this piece of code
- **except** : if any exceptions occur, run this piece of code. There can be several **except** clauses for different types of error, although only one except clause will be executed at any time.
- **else** : run this piece of code regardless of whether an exception occurred or not. The **else** clause is optional.

In this activity you will write code that causes some different types of error, and modify the code to handle the exceptions.

47.2 Create a project

Create a project called **exceptionsProject** with a **main.py**.

47.3 Write a script that works

Add this code to the script:

```
l = [0, 1, 2, 3, 4]

for i in range(0, 5):
    print(l[i])

print("Finished")
```

Run the script. The output will look like this:

```
0
1
2
3
4
Finished
```

47.4 Break the script

Change the 5 in the **for** loop to a 6.

```
for i in range(0, 6):
```

Run the script again. This time the script crashes out:

```
0
1
2
3
4
Traceback (most recent call last):
  File "C:\python-workspace\exceptionsProject\main.py", line 4, in <module>
    print(l[i])
IndexError: list index out of range
```

The problem is that the list has five items, but the loop is trying to run six times.

Of course we could simply change the loop back to 5 again. But imagine this is part of a larger program where the range comes from another part of the program- it might not be so simple to fix.

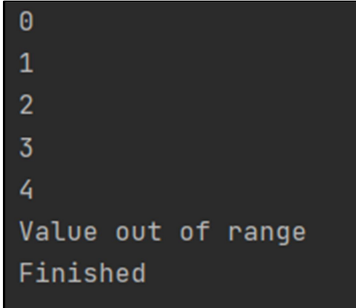
47.5 Fix the script with exception handling

Instead we will fix the program with some exception handling.

Modify the loop code to look like this:

```
try:
    for i in range(0, 6):
        print(l[i])
except:
    print("Value out of range")
```

Run the script again. This time the error still occurs, but the script catches it, and is able to proceed.



```
0
1
2
3
4
Value out of range
Finished
```

47.6 Where to put try...except?

Note that in the script, when the exception occurs it runs the **except** clause then moves on to the code outside the **try...except** block.

To see this, change the range in the **for** loop to have an upper limit of 9:

```
for i in range(0, 9):
```

Run the script. Note that the **Finished** appears after the loop runs out of values in the list- the loop never goes past that:

```
0
1
2
3
4
Value out of range
Finished
```

Modify the block to look like this, with the try...catch inside the loop:

```
for i in range(0, 9):
    try:
        print(l[i])
    except:
        print("value out of range")
print("Finished")
```

Run again. This time, the loop executes for all values in the range:

```
0
1
2
3
4
Value out of range
Value out of range
Value out of range
Value out of range
Finished
```

Note that PEP-8, the Python style guide, suggests not using a “bare” except statement- one that does not have an exception type stated- it complains that the command is “too broad” (in Pycharm this will show a warning). It is better to specify the expected exception types in the command. In this example, the exception type is **IndexError**- this is shown in the original error message. Add the exception type to the command as follows:

```
except IndexError:
```

The warning message disappears.

47.7 Multiple exception types

In one try block it is possible to specify multiple different exception types.

Modify the code so that it writes the values out to a file- modify the try block to look like this:

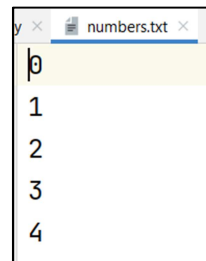
```
try:
    outfile = open("numbers.txt.", "w")
    for i in range(0, 9):
        print(l[i], file=outfile)
```

```
outfile.close()

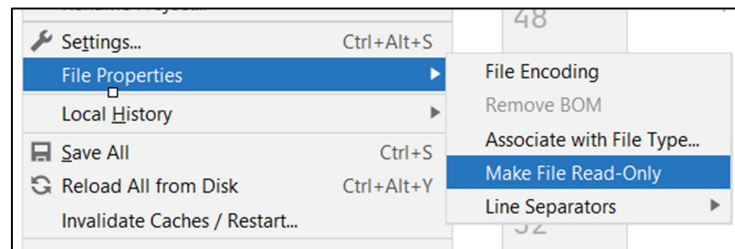
except IndexError:
    print("Value out of range")

print("Finished")
```

Run the program. The index error exception will be raised but also the numbers.txt file will be created- it is created in the same folder as the Python project. Open the file and make sure it contains the expected values:



To cause an exception, mark the file as read-only. In PyCharm, this can be done by selecting the file in the project and choosing File>File Properties>Mark File Read Only from the menu.



If you are not using PyCharm, find the file in the operating system file browser and mark it as read-only.

Run the program again. This time an exception occurs.

```
Traceback (most recent call last):  
  File "C:\python-workspace\workbook - exceptionsProject\main.py", line 48, in <module>  
    outfile = open("numbers.txt.", "w")  
PermissionError: [Errno 13] Permission denied: 'numbers.txt.'
```

To avoid the program crashing out, add an extra except clause to the try block:

```
except PermissionError:  
    print("You are not allowed to write to the file")
```

Run the program again and check that the exception is raised.

```
You are not allowed to write to the file  
Finished
```

47.8 Explore more

47.8.1 Other exception types

There are lots of potential exceptions that can be raised by a program. Try adding these lines to the code and see what exception types are raised:

```
n = 5 / 0  
n = 5 + "hello"
```

Add appropriate **except** clauses for these exceptions.

Note that even though PEP-8 disapproves of it, it is still possible to add a generic **except** clause to catch exceptions that are not specifically handled.

47.8.2 User defined exceptions

It is possible for a user to define their own exceptions, and raise them if problems occur in their code. Refer to the Python documentation about error handling for more information: <https://docs.python.org/3/tutorial/errors.html>