

Python Programming Section 8 – File I/O

**UEE60411 Advanced Diploma of Computer Systems Engineering**

UEENEED103A Evaluate and modify Object Oriented code programs

For further information:

Steve Gale

ICT – Science and Technology Ph: (03) 5225 0940

Email: [sgale@gordontafe.edu.au](mailto:sgale@gordontafe.edu.au)

# Exceptions

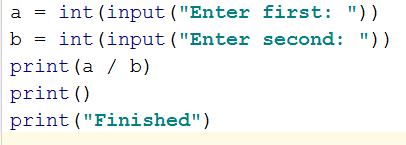
One thing that we can often find when trying to open a text file in Python is that if the file doesn’t exist, the program **throws an exception**. This means that it generates a run-time error, and the program crashes.

In most cases of a program error, we would simply debug and repair the incorrect code. However, this kind of exception doesn’t result from incorrect code – it happens because a file doesn’t exist. The code itself is correct, and there is no way of predicting in advance the absence of a file.

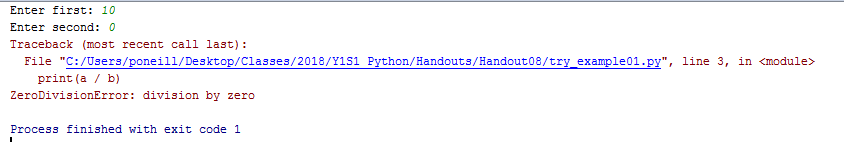
So we must set up an **error handler** which will prevent the program from crashing if something goes wrong. We do that using a **try-except** block.

Let’s create a program that will perform a simple math problem: dividing one input number by a second input number and printing out the result. It’s a simple program – however, if the *second* input number (the **divisor**) is 0, the program will crash, as division by zero is impossible.

Try it – enter the following program and run it, first with 10 and 2 as the inputs (giving 5 as the result), then try the same with 0 as the second input.

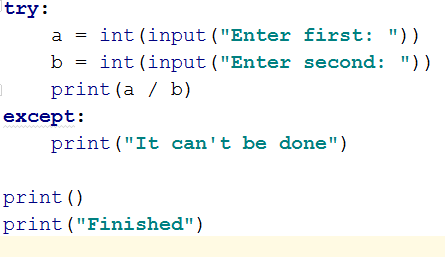


When you enter 0 as the second value, you will get an error:

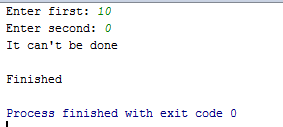


We can fix this by adding a **try-except** block, which allows the program to **try** something, and then perform some other action if the attempt generates an **exception**.

To do the same thing as above, but with the ability to catch the error, modify the program as follows:



The output for this program, if we deliberately create an exception by using 0 as our second input, will produce this:

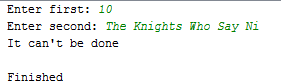


This is much more elegant, doesn’t crash the program, and can actually provide useful feedback for the user, rather than a crash.

You can (and usually should) include *multiple* **except** clauses which will provide useful feedback to the user, explaining exactly what has gone wrong.

With the above program, if we enter “The Knights Who Say Ni” as the second input, we will get

*exactly* the same result as entering 0. Try it.



This is because the **except** clause just triggers if there is an error, *of any kind*. We can, however, add more

**except** clauses which can be used to tailor the response, depending on what sort of error was generated.

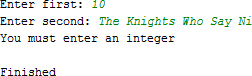
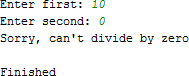
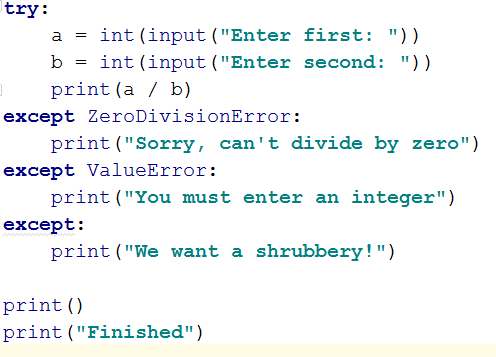
For example, we can use the following code to modify our program, and then try putting in 0 as the second value, and then “The Knights Who Say Ni” again.

14/04/2020

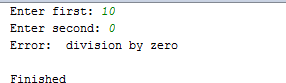
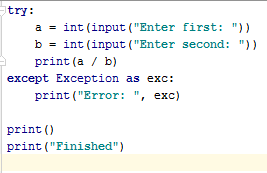
Business and ICT

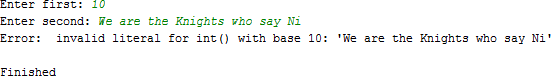
Page **3** of **15**

Python HO 08.docx



The last except clause is a “fall-through”. If an error occurs, and it is (in this case) neither a

**ZeroDivisionError** nor a **ValueError**, the “We want a shrubbery!” outcome will occur. We can also use the following code to display a system explanation of what has gone wrong.



Pa Python HO 08.d

# Writing to external files.

Python is able to read, write and append external files.

|  |  |
| --- | --- |
| Mode | Description |
| 'r' | Open a file for reading. (default) |
| 'w' | Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists. |
| 'x' | Open a file for exclusive creation. If the file already exists, the operation fails. |
| 'a' | Open for appending at the end of the file without truncating it. Creates a new file if it does not exist. |
| 't' | Open in text mode. (default) |
| 'b' | Open in binary mode. |
| '+' | Open a file for updating (reading and writing) |

The simplest file type to work with is the text file.

Here is some sample code below that writes, and reads and appends a text file called testfile.txt.



Python Programming

14/04/2020

Business and ICT

ge **5** of **15**

# make new textfile

# "w" is write, "r" is read, "a+" is append file = open("testfile2.txt","w")

#write to the file file.write("1. New file?") file.write("\n2. Second line ") file.write("\n3. Still more")

file.write("\n4. The new line statements work in the text file.")

file.write("\n5. You can read it and print it out as well")

#read from the file and print file =open("testfile2.txt","r")

#read line 1 print(file.readline())

#print a set no of letters print(file.read(16))

#read the rest of the contents of testfile.txt print(file.read())

file.close()

print("\nOpen the file in append mode and add a line")

#append

file = open("testfile2.txt","a+") file.write("\n6. Appended line ")

#change to read mode.

file= open("testfile2.txt","r") print(file.read())

## Output:

1. New file?
2. Second line
3. Still more
4. The new line statements work in the text file.
5. You can read it and print it out as well

Open the file in append mode and add a line

1. New file?
2. Second line
3. Still more
4. The new line statements work in the text file.
5. You can read it and print it out as well
6. Appended line

If you open the testfile.txt, you can see what has been written to it.

# 

# File I/O Exercises

1. Write instructions to an external file called **commands.txt**

* list the seven modes that we can use for reading and writing to external files from python (see table above)
* list each item on a separate line.
* Append on a new line and simply write “Appended line”.
* Print out all the content from the text file.
* Close the connection to the file.
* Save your python file as **commands.py**

1. **File\_ex02.py** collect details relating to employees (Hint use a loop for no. of employees) only 2 records are shown for brevity, but you should be using 5

#### Part2 – Print the details out like below to a text file.

