Python 9

Exceptions

There are a few different types of errors when coding. Syntax errors, logic errors, and exceptions. You have probably encountered all three. Syntax and logic errors are issues you need to deal with while coding. An exception is a special type of error that can be informative and used to write code to respond to this type of error. This is especially relavent when dealing with user input. What if they don't give you any, or it is the wrong kind of input. We want our code to be able to detect these types of errors and respond accordingly.

```
#!/usr/bin/env python3

import sys
file = sys.argv[1]

print("User provided file:" , file)
```

> This code takes user provided input and prints it

Run it.

```
$ python scripts/exceptions.py test.txt
User provided file: test.txt
```

What happens if the user does not provide any input and we try to print it?

```
$ python scripts/exceptions.py
Traceback (most recent call last):
  File "scripts/exceptions.py", line 4, in <module>
    file = sys.argv[1]
IndexError: list index out of range
```

> We get an **IndexError** exception, which is raised when an index is not found in a sequence.

We have already seen quite a few exceptions throughout the lecture notes, here are some:

- ValueError: math domain error
- AttributeError: 'list' object has no attribute 'rstrip'
- SyntaxError: EOL while scanning string literal
- NameError: name 'GGTCTAC' is not defined

- SyntaxError: Missing parentheses in call to 'print'
- AttributeError: 'int' object has no attribute 'lower'
- IndexError: list assignment index out of range
- NameError: name 'HDAC' is not defined

Link to Python Documentation of built in types of exceptions

We can use the exception to our advantage to help the people who are running the script. We can use a try/except condition like an if/else block to look for exceptions and to execute specific code if we **do not have** an exception and do something different if we **do have** an exception.

```
#!/usr/bin/env python3
import sys

file = ''
try:
  file = sys.argv[1]
  print("User provided file:" , file)
except:
  print("Please provide a file name")
```

> We need to "try" to get a user provided argument. If we are successful then we can print it out. If we try and fail, we execute the code in the except portion of our try/except and print that we need a file name.

Let's run it WITH user input

```
$ python3 scripts/exceptions_try.py test.txt
User provided file: test.txt
```

> It runs as expected

Let's run it WITHOUT user input

```
$ python scripts/exceptions_try.py
Please provide a file name
```

> Yeah, the user is informed that they need to provide a file name to the script

What if the user provides input but it is not a valid file or the path is incorrect? Or if you want to check to see if the user provided input as well as if it can open the input.

We can add multiple exception tests, like if/elif block. Each except statement can specify what kind of exception it is waiting to recieve. If that kind of exception occures, that block of code will be executed.

```
import sys

file = ''
try:
  file = sys.argv[1]
  print("User provided file name:" , file)
  FASTA = open(file, "r")
  for line in FASTA:
    line = line.rstrip()
    print(line)
except IndexError:
  print("Please provide a file name")
except IOError:
  print("Can't find file:" , file)
```

> Here we test for an IndexError: Raised when an index is not found in a sequence.
> The IndexError occurs when we try to access a list element that does not exists.
> And we test for a IOError: Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
> The IOError happens when we try to access a file that does not exist.

Let's run it with a file that does not exist.

```
$ python scripts/exceptions_try_files.py test.txt
User provided file name: test.txt
Can't find file: test.txt
```

> This informs the user that they did provide input but that the file listed can not be found.

Let's run it with no input

```
$ python scripts/exceptions_try_files.py
Please provide a file name
```

> This informs the user that they need to provide a file.

try/except/else/finally

Lets summarize what we have covered and add on `else` and `finally`.

```
try:
  # try block is executed until an exception is raised
except _ExceptionType_:
  # if there is an exception of "ExceptionType" this block will be executed
  # there can be more than one except block, just like an elif
except:
  # if there are any exceptions that are not of "ExceptionType" this except block will
be executed
else:
  # the else block is executed after the try block has been completed, which means
there were no exceptions raised
finally:
  # the finally block is executed if exceptions are or are not raised (no matter what
happens)
```

Getting more information about an exception

Some exceptions can be thrown for multiple reasons, for example, ErrorIO will occur if the file does not exist as well as if you don't have permissions to read it. We can get more information by viewing the contents of our Exception Object. Yes, an exception is an object too! The system errors get stored in the exception object. To access the object use `as` and supply a variable name, like 'ex'

```
file = ''
try:
  file = sys.argv[1]
  print("User provided file name:" , file)
  FASTA = open(file, "r")
  for line in FASTA:
    line = line.rstrip()
    print(line)
except IndexError:
  print("Please provide a file name")
except IOError as ex:
  print("Can't find file:" , file , ': ' , ex.strerror  )
```

> Here we added `except IOError as ex` and now we can get the 'strerror' message from ex.

Run it.

```
$ python scripts/exceptions_try_files_as.py  test.txt
User provided file name: test.txt
Can't find file: test.txt :  No such file or directory
```

> Now we know that this file name or path is not valid

Raising an Exception

We can call or raise exceptions too!! This is accomplished by using a `raise` statement.

1. First, create a new Exception Object, i.e., `ValueError()`
2. Use the Exception Object in a Raise statment `raise ValueError('your message')`

Let's raise an exception if the file name does not end in 'fa'

```
import sys

file = ''
try:
  file = sys.argv[1]
  print("User provided file name:" , file)
  if not file.endswith('.fa'):
    raise ValueError("Not a FASTA file")
  FASTA = open(file, "r")
  for line in FASTA:
    print(line)
except IndexError:
  print("Please provide a file name")
except IOError as ex:
  print("Can't find file:" , file , ': ' , ex.strerror  )
```

> Here we raise a known exception, 'ValueError', if the file does not end with (uses `endswith()` method).

Let's run it.

```
$ python scripts/exceptions_try_files_raise.py test.txt
User provided file name: test.txt
Traceback (most recent call last):
  File "scripts/exceptions_try_files_raise.py", line 10, in <module>
    raise ValueError("Not a FASTA file")
ValueError: Not a FASTA file
```

> Our exception get's raised, now lets do something with it.

```
import sys

file = ''
try:
  file = sys.argv[1]
  print("User provided file name:" , file)
  if not file.endswith('.fa'):
    raise ValueError("Not a FASTA file")
  FASTA = open(file, "r")
  for line in FASTA:
    print(line)
except IndexError:
  print("Please provide a file name")
except ValueError:
  print("File needs to be a FASTA file and end with .fa")
except IOError as ex:
  print("Can't find file:" , file , ': ' , ex.strerror  )
```

> Here we created an exception to catch any ValueError

Let's Run it.

```
$ python scripts/exceptions_try_files_raise_value.py test.txt
User provided file name: test.txt
File needs to be a FASTA file and end with .fa
```

> We get a great error message now.

But what if there is another ValueError, how can we tell if has anything to do with the FASTA file extension or not? Answer: the message will be different.

Creating Custom Exceptions

We can create our own custom exception. We will need to create a new class of exception. Below is the syntax to do this.

```
import sys

class NotFASTAError(Exception):
  pass


file = ''
try:
  file = sys.argv[1]
  print("User provided file name:" , file)
  if not file.endswith('.fa'):
    raise NotFASTAError("Not a FASTA file")
  FASTA = open(file, "r")
  for line in FASTA:
    print(line)
except IndexError:
  print("Please provide a file name")
except NotFASTAError:
  print("File needs to be a FASTA file and end with .fa")
except IOError as ex:
  print("Can't find file:" , file , ': ' , ex.strerror  )
```

> Here we created a new class of exception called 'NotFASTAError'. Then we raised this new exception.

Let's Run it.

```
$ python scripts/exceptions_try_files_raise_try.py test.txt
User provided file name: test.txt
File needs to be a FASTA file and end with .fa
```

> Our new class of exception, NotFASTAError, works just like the built in exceptions.

Link to Python 9 Problem Set