Bioinformatical problem solving with Python

Wednesdays 17:30-19:00, M801

alexander.nater@uni-konstanz.de

Not covered topics so far

- **Error handling (exceptions)**
- Numerical and scientific libraries (SciPy, NumPy, pandas, etc.)
- Plotting libraries (Matplotlib)
- Software design

Exercise

Write a function that takes a list of numbers and returns the mean. Write a script that calls the function with an empty list. What happens? How could you prevent program termination?

Usage example

```
def get_mean(numbers):
    return sum(numbers) / len(numbers)

my_list = [1, 2, 3]
print(get_mean(la))
> 2.0

my_list = []
print(get_mean(lb))
> ZeroDivisionError: division by zero

def get_mean(numbers):
    if not numbers:
        return 0
    else:
        return sum(numbers) / len(numbers)
```

- Python uses exceptions to signal errors to the user.
- Exceptions are instances of the Exception base class or its subclasses.
- Custom-made exceptions should inherit from the Exception base class.
- Exceptions are raised with the ‘raise’ keyword followed by an Exception instance.

Usage example

```
def get_mean(numbers):
    if not numbers:
        raise Exception("The list of numbers is empty!")
    return sum(numbers) / len(numbers)
```

Exception handling

- Python raises an ‘ZeroDivisionError’ if we try to divide a number by zero.
- This error is propagated to the built-in error handler, which prints the error message and terminates the program.
- Instead of rewriting our function, we can just try to intercept and handle exceptions on our own.
- A try-except block is used to check for the occurrence of any exceptions in the executed code.

```
try:
    mean = get_mean(my_list)
except Exception as exp:
    print(“An exception has been raised!”)
    print(“The error type is: {}”.format(type(exp)) )
    print(“The error message is: {}”.format(exp) )
else:
    print(“No exception occurred.”)
    print(“The result is: {}”.format(mean) )
finally:
    print(“This code is executed no matter what.”)
```

Custom exceptions

- Custom-made exceptions should inherit from the Exception base class.
- The subclass should be named after the error type, e.g. ZeroDivisionError, NameError, TypeError, etc.
- The docstring should be used to document the type of error.

Example

```
class TooManyError(Exception):
    """Error indicating that the list contains too many numbers."""
    pass

def get_mean(numbers, limit = 100):
    if len(numbers) > limit:
        raise TooManyError("Too many numbers in list!")
    return sum(numbers) / len(numbers)
```

Example

```
my_list = list(range(1000))
try:
    mean = get_mean(my_list)
except TooManyError as exp:
    print("A TooManyError has been raised!")
    print("The error message is: {}".format(exp) )
except ZeroDivisionError as exp:
    print ("A ZeroDivisionError has been raised!")
    print("The error message is: {}".format(exp) )
except Exception as exp:
    print("A {} has been raised".format(type(exp)) )
    print("The error message is: {}".format(exp) )
else:
    print("No exception occurred.")
    print("The result is: {}".format(mean) )
```

The os.mkdir() function raises an exception if asked to create a directory that already exists. Write a mkdir_p(path) function that creates a directory at path, but does nothing if the directory already exists. Return True if the directory has been actually created, and False if nothing was changed on the file system.