Exceptions

Checking Input Variables
Avoid run-time errors

Input error

When prompted for input from the console, the user may enter an incorrect value or format.

```
x = input("Please enter an integer: ")
return int(x)
```

Output

```
Please enter an integer:  a
ValueError: invalid literal for int() with base 10: 'a'
```

Exception handling

To prevent the program from crashing, critical code can be wrapped into a `try ... except` construct

```
while True:
    try:
        v = input("Please enter an integer: ")
        x = int(v)
    except ValueError:
        print("Invalid input, please try again")
    else:
        return x
```

The input is repeated until the user enters an integer.

Output

```
Please enter an integer: a
Invalid input, please try again

Please enter an integer: b
Invalid input, please try again
```

Exception handling

Embedded in a while loop

```
while True:
    try:
        v = input("Please enter an integer: ")
        x = int(v)
    except ValueError:
        print("Invalid input, please try again")
    else:
        return x
```

Output

```
Please enter an integer: a
invalid input, please try again

Please enter an integer: 1

Please enter an integer: b
invalid input, please try again

Please enter an integer: 2
Sum: 3
```

Exception handling

Read in floating point numbers

```
while True:
        try:
            v = input("Please enter a floating point number: ")
            x = float(v)
        except ValueError:
            print("Invalid input, please try again")
        else:
            return x
```

Output

```
Please enter a floating point number: a
Invalid input, please try again

Please enter a floating point number: 2.4
```

General construct

```
try:
    # code which might throw an exception

except:
    # exception handling, print error message

else:
    # continue if no exception was thrown
```

Error types

Python supports different types of errors

- ValueError -> Type conversion incorrect (or impossible)
- TypeError -> Function call with incorrect data type
- ZeroDivisionError -> Attempt to divide by 0
- IndexError -> Attempt to access a non-existent list item
- . . .

Example with different error types

```
z = input("Please insert a number: ")
try:
  y = float(z)
  d = x / y
except ValueError:
  print("ValueError: Invalid input", z)
  return None
except TypeError:
  print("TypeError: Invalid argument", x)
  return None
except ZeroDivisionError:
  print("ZeroDivisionError: Invalid divisor", z)
  return None
else:
  print(x,"/",y,"=",d)
  return d
```

Output

```
Please insert a number: 1
4 / 1.0 = 4.0

Please insert a number: 0
ZeroDivisionError: Invalid divisor 0
```

Index Error in Lists

If an attempt is made to access a non-existent position of a list, an IndexError is generated and the procedure aborts.

```
numbers = [1,2,3,4]
for i in range(5):
    print(numbers[i])
```

Output

```
1
2
3
4
Traceback (most recent call last):

  File "C:\Temp\PythonBsp\Exceptions_Files\ExceBspe.py", line 59, in <module>
    print(numbers[i])

IndexError: list index out of range
```

A dictionary does not throw an error.

```
squares = {1:1, 3:9, 4:16, 6:36}
for i in range(1,5):
    print(squares.get(i))
```

Output

```
1
None
9
16
```

Summarize Exceptions

The except statement can catch multiple exceptions at the same time. The following list contains integers, letters, and complex numbers. The various errors can be handled at the same time.

```
listValue = [1, 'a', 2, 3j, 5]
for i in listValue:
    try:
        print(int(i) % 2)
    except (ValueError, TypeError):  ///  To kind of errors
        print("Invalid value in list: " + str(i))
```

Output

```
1
Invalid value in list: a
0
Invalid value in list: 3j
1
```

Berner Fachhochschule

Skip invalid input

Sometimes it is also possible to ignore mistakes and simply continue in the process.
The key word for this is **pass**

```
listValue = [1, 'a', 2, "B", 6]
for i in listValue:
    try:
        print(int(i) % 2)
    except (ValueError, TypeError):
        pass
```

Output

```
1
0
0
```

Check correct type

Sometimes we want to make sure that a variable has the correct type. This can be useful, to avoid application errors:

```
class Person():
  def __init__(self, firstname, lastname):
        . . .
  def getFirstname(self):
    return self.__firstname
```

```
if isinstance(p, Person):
    return p.getFirstname()
else:
    raise TypeError("p is no person")
```