# COMP1531

## 2.6 - Python - Exceptions

# Python - Exceptions

An **exception** is an action that disrupts the normal flow of a program. This action is often representative of an error being thrown. Exceptions are ways that we can elegantly recover from errors

# Python - Exceptions

The simplest way to deal with problems...

**Just crash**

*exception_1.py*

```
1  import sys
2
3  def sqrt(x):
4      if x < 0:
5          sys.stderr.write("Error Input < 0\n")
6          sys.exit(1)
7      return x**0.5
8
9  if __name__ == '__main__':
10     print("Please enter a number: ",)
11     inputNum = int(sys.stdin.readline())
12     print(sqrt(inputNum))
```

# Python - Exceptions

Now instead, let's raise an exception

However, this just gives us more information, and doesn't help us handle it

*exception_2.py*

```
1  import sys
2
3  def sqrt(x):
4      if x < 0:
5          raise Exception(f"Error, sqrt input {x} < 0")
6      return x**0.5
7
8  if __name__ == '__main__':
9      print("Please enter a number: ",)
10     inputNum = int(sys.stdin.readline())
11     print(sqrt(inputNum))
```

# Python - Exceptions

If we catch the exception, we can better handle it

*exception_3.py*

```
1  import sys
2
3  def sqrt(x):
4      if x < 0:
5          raise Exception(f"Error, sqrt input {x} < 0")
6      return x**0.5
7
8  if __name__ == '__main__':
9      try:
10         print("Please enter a number: ",)
11         inputNum = int(sys.stdin.readline())
12         print(sqrt(inputNum))
13     except Exception as e:
14         print(f"Error when inputting! {e}. Please try again:")
15         inputNum = int(sys.stdin.readline())
16         print(sqrt(inputNum))
```

# Python - Exceptions

Or we could make this even more robust

*exception_4.py*

```
1  import sys
2
3  def sqrt(x):
4      if x < 0:
5          raise Exception(f"Error, sqrt input {x} < 0")
6      return x**0.5
7
8  if __name__ == '__main__':
9      print("Please enter a number: ",)
10     while True:
11         try:
12             inputNum = int(sys.stdin.readline())
13             print(sqrt(inputNum))
14             break
15         except Exception as e:
16             print(f"Error when inputting! {e}. Please try again:")
```

# Python - Exceptions

Key points:

- Exceptions carry data
- When exceptions are thrown, normal code execution stops

*throw_catch.py*

```python
import sys

def sqrt(x):
    if x < 0:
        raise Exception(f"Input {x} is less than 0. Cannot sqrt a number < 0")
    return x**0.5

if __name__ == '__main__':
    if len(sys.argv) == 2:
        try:
            print(sqrt(int(sys.argv[1])))
        except Exception as e:
            print(f"Got an error: {e}")
```

# Python - Exceptions

Examples with pytest (very important for project)

*pytest_except_1.py*

```python
import pytest

def sqrt(x):
    if x < 0:
        raise Exception(f"Input {x} is less than 0. Cannot sqrt a number < 0")
    return x**0.5

def test_sqrt_ok():
    assert sqrt(1) == 1
    assert sqrt(4) == 2
    assert sqrt(9) == 3
    assert sqrt(16) == 4

def test_sqrt_bad():
    with pytest.raises(Exception, match=r"*Cannot sqrt*"):
        sqrt(-1)
        sqrt(-2)
        sqrt(-3)
        sqrt(-4)
        sqrt(-5)
```

# Python - Exception Sub-types

Other basic exceptions can be caught
with the "Exception" type

*pytest_except_2.py*

```
1  import pytest
2
3  def sqrt(x):
4      if x < 0:
5          raise ValueError(f"Input {x} is less than 0. Cannot sqrt a number < 0")
6      return x**0.5
7
8  def test_sqrt_ok():
9      assert sqrt(1) == 1
10     assert sqrt(4) == 2
11     assert sqrt(9) == 3
12     assert sqrt(16) == 4
13
14 def test_sqrt_bad():
15     with pytest.raises(Exception):
16         sqrt(-1)
17         sqrt(-2)
18         sqrt(-3)
19         sqrt(-4)
20         sqrt(-5)
```