COMP10001 Foundations of Computing
Project 1 Review; Exceptions and Assertions

Semester 1, 2019
Tim Baldwin, Nic Geard, Farah Khan, and Marion Zalk

THE UNIVERSITY OF MELBOURNE

— VERSION: 1537, DATE: APRIL 18, 2019 —

Reminders

- Project 2 due Thursday 9/5
- No Grok worksheets due next week!
- Grok worksheets 14 & 15 due Monday 13/5
- Revision lecture this Friday 3 May

Lecture Agenda

- Last lecture:
  - Modules
  - List comprehensions
  - File IO
- This lecture:
  - Project 1 Review
  - Exception handling
  - Assertions

Lecture Outline

1 Exception handling

2 Assertions

Exception Handling

- Python prints the `Exception` causing the error:

```
>>> 9/0
Traceback (most recent call last):
  File "<web session>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero

>>> 1 + "2"
Traceback (most recent call last):
  File "<web session>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'

>>> 1 + i
Traceback (most recent call last):
  File "<web session>", line 1, in <module>
NameError: name 'i' is not defined
```

Exception Handling

- Other common run-time exceptions are:
  - `IndexError`: raised when an index is out of range
  - `KeyError`: raised when a key is not found in a dictionary
- It is possible to "handle" exceptions within your code using `try: ... except Exception: ....`
- `try` attempts to execute its block of code, and passes off to the exception handlers (which are also tested in linear order) only if an exception is raised during the execution, before running the code block attached to `finally`

Exception Handling

For example (does not work as expected):

```
x = "not a number"
while type(x) != int:
    x = int(input("Please enter a number: "))
```

Exception Handling

For example (does not work as expected):

```
x = "not a number"
while type(x) != int:
    x = int(input("Please enter a number: "))
```

Fixed:

```
x = "not a number"
while type(x) != int:
    try:
        x = int(input("Please enter a number: "))
    except ValueError:
        print("Oops! Try again...")
```

Exception Handling

- You can catch/handle more than one exception

```
"""Print recipricol of input integer."""
while True:
    try:
        x = input("Please enter an integer: ")
        print(1/int(x))
    except ValueError:
        print("That's not an integer.")
    except ZeroDivisionError:
        print("Cannot find recipricol of 0")
```

Exception Handling

- It is considered best practice to only catch errors that you are interested in.

```
while True:
    try:
        x = input("Please enter an integer: ")
    except:
        print("Something went wrong... who knows what?")
```

- `try` and `except` should be used for ‘exceptional’ circumstances.

Lecture Outline

Assertions

- To date, we have tended to assume well-behaved inputs to our functions etc., and lived with the fact that ill-behaved inputs will cause a logic or run-time error, e.g.:

```
def withdraw(amount,balance):
    if balance < -100:
        print("Insufficient balance")
        return(balance)
    else:
        print("Withdrawn")
        return(balance - amount)
>>> withdraw(100,False)
Withdrawn
-100
```

Assertions

- One way to ensure that the inputs are of the right type is with `assert`:

```
def withdraw(amount,balance):
    assert type(balance) == int
    if balance < -100:
        print("Insufficient balance")
        return(balance)
    else:
        print("Withdrawn")
        return(balance - amount)
>>> withdraw(100,'a')
Traceback (most recent call last):
...
AssertionError
```

Assertions

- Note, however, that assertions should be used sparingly and reserved for “impossible” code states
- Use an explicit `if` statement if the result is important to the logic of the code
- Assertions are an important tool in defensive programming

Lecture Summary

- The call stack is your friend
- Namespaces and scope
- Exceptions: dealing with problems gracefully

Lecture Summary

- How can we handle runtime errors gracefully?
- Exceptions: ask forgiveness not permission!
- How can we ensure that certain conditions are met?
- Assertions: raise an error if things are not as they should be.