Type Conversion – Most Common Ones

Function	**Description**
int(x [,base])	Converts x to an integer base specifies the base if x is a string.
long(x [,base])	Converts x to a long integer base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
hex(x)	Converts an integer to a hexadecimal string.
str(x)	Converts object x to a string representation.
chr(x)	Converts an integer to a character.
complex(real [,imag])	Creates a complex number.

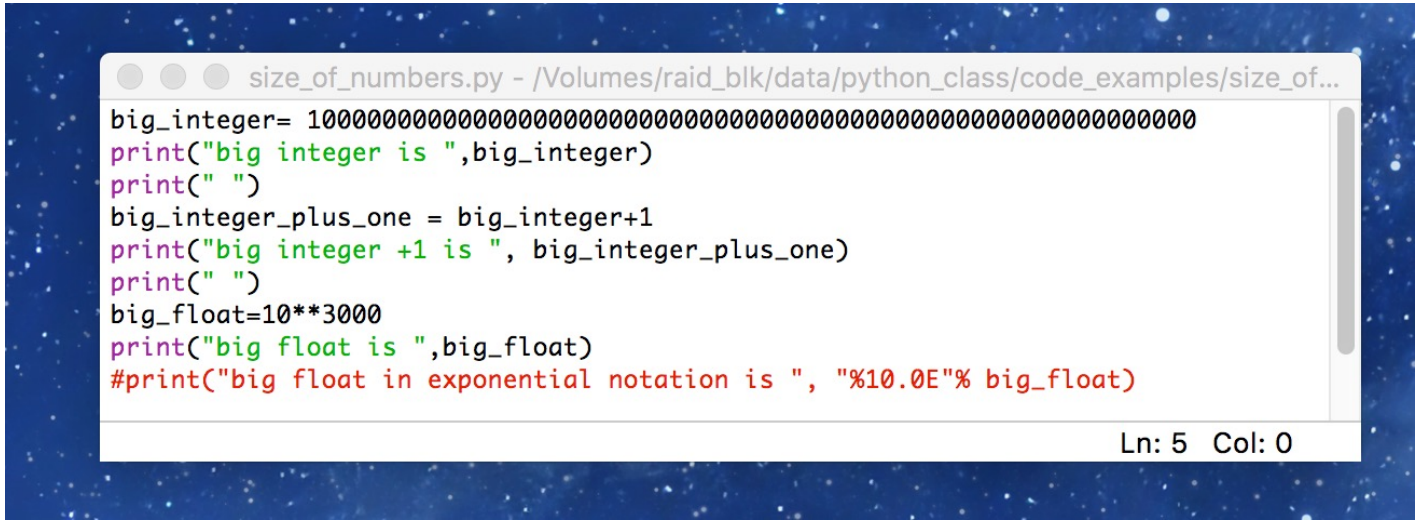
Type Conversion – Less Common Ones

Function	**Description**
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary d must be a sequence of (key,value) tuples.
frozenset(s)	Converts s to a frozen set.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
oct(x)	Converts an integer to an octal string.

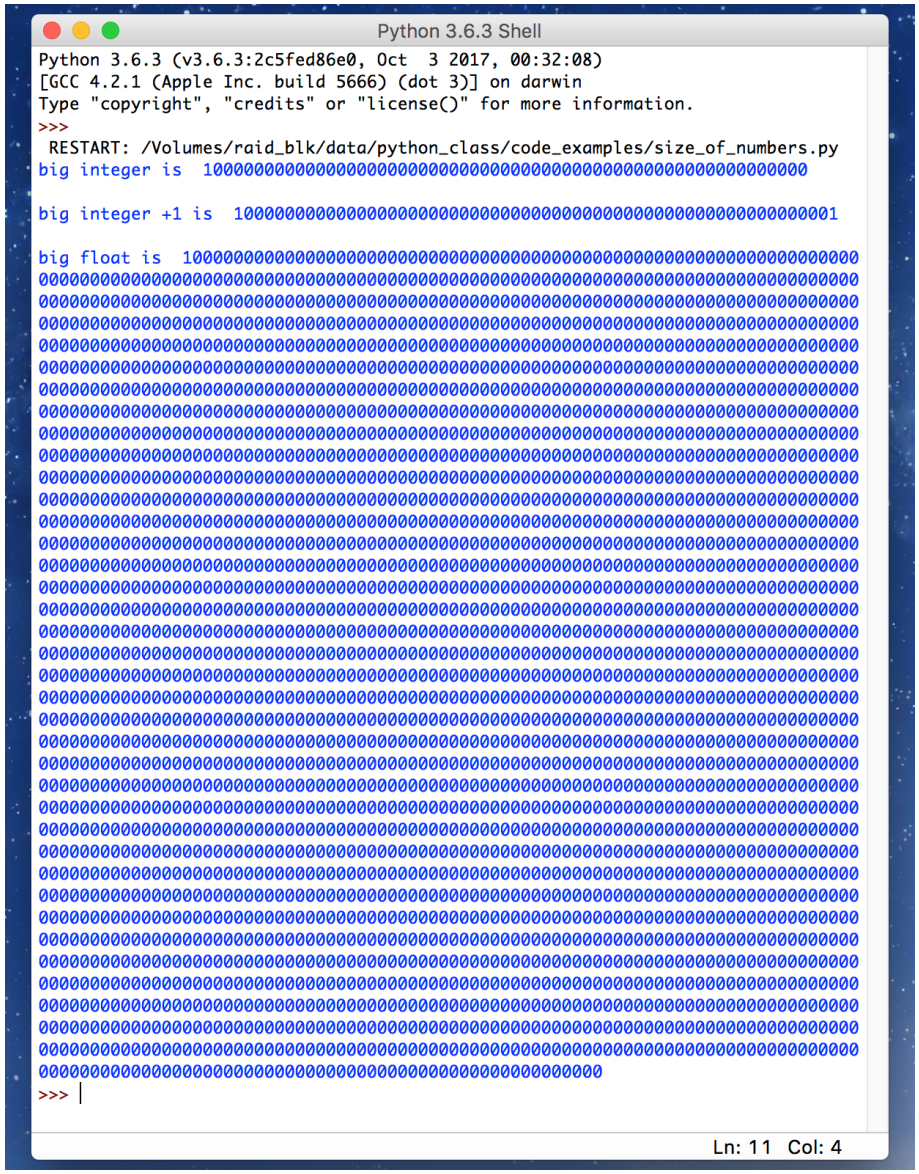
Size of numbers

- Unlike other programming languages
- Python has no real limit on the size of numbers
- In extreme cases, you may run out of memory
- However an there are limits on how number can be displayed or converted

Size of numbers example



Output of Size of numbers Example



Exception coding

- When we have large programs, we may need to jump out of pieces of the program
- When we encounter an error, the program will terminate, i.e., stop running
- If we can capture the error we may be able to fix the problem and continue running the program
- Example: I build a calculator. I ask for input of a number. The person enters "z". The program crashes/terminates/stops running
- If I capture the wrong input, I can ask the user to enter the correct type of data

Exception Coding

- try/except/else
- try/except/finally
- raise
- assert

try/except/else

- The try statement works as follows.
  - First, the try clause (the statement(s) between the try and except keywords) is executed.
    - If no exception occurs, the except clause is skipped and execution of the try statement is finished.
    - If an exception occurs during execution of the try clause, the rest of the clause is skipped.
  - Then if its type matches the exception named after the except keyword, the except clause is executed, and then execution continues after the try statement.
  - If an exception occurs which does not match the exception named in the except clause, it is passed on to outer try statements;
- If no handler is found, it is an unhandled exception and execution stops with a message as shown above.
- A try statement may have more than one except clause, to specify handlers for different exceptions.
- At most one handler will be executed. Handlers only handle exceptions that occur in the corresponding try clause, not in other handlers of the same try statement.

try/except/finally

- This set of statements works similar to the try/except/else
- The "finally" statement is always run for no exceptions and handled exceptions
- It's best use it to make the code clearer
- It is not required

raise

- When testing code, it may not be possible to determine how an exception will occur
- The raise statement is used to force an exception for testing
- It should be removed after testing
- It can also be added around a statement the only get executed when the debug mode is enabled
  - See python documentation for debug mode
  - https://docs.python.org/3.2/library/pdb.html

assert

- The goal of an assertion in Python is to inform developers about unrecoverable errors in a program.
- Python's assert statement is a debugging aid, not a mechanism for handling run-time errors.
- The goal of using assertions is to let developers find the likely root cause of a bug more quickly.
- An assertion error should never be raised unless there's a bug in your program
- Basically the approach is to make (force) an error to test whether or not the error is handled correctly

Example code for exceptions

exceptions.py - /Users/joe/Documents/exceptions.py (3.6.3)

```
"""
This code presents examples for serveral types of exception handling.
The goal is to keep the code running smoothly in the presence of problems
like poor user input
unknown errors
"""

# try except example
for i in range(1,3):
    try:
        y= input("Please enter a number: ")
        x = int(y)
        print("thank you, that was a valid number of value: ",x)
    except ValueError:
        print("Oops!  That was not a valid number. You entered \" ", y,  "\" Try again...")
    print(" ")

# try except finally example
for i in range(1,3):
    x= input("Please enter the numerator  : ")
    x=int(x)
    y= input("Please enter the denominator: ")
    y=int(y)
    try:
        result = x/y
        print("The result of the division is ",result)
    except ZeroDivisionError:
        print("Oops! dividing by zero Try again...")
    finally:
        print("finally runs whether there is an exception or no exception")
    print(" ")

# assert example
for i in range(1,2):
    try:
        y= input("Please enter a number: ")
        x = int(y)
        assert int("a")
        print("thank you, that was a valid number of value: ",x)
    except ValueError:
        print("Oops!  That was not a valid number. You entered \" ", y,  "\" Try again...")
    print(" ")

# try except finally example with raise
for i in range(1,2):
    x= input("Please enter the numerator  : ")
    x=int(x)
    y= input("Please enter the denominator: ")
    y=int(y)
    try:
        result = x/y
        raise ZeroDivisionError
        print("The result of the division is ",result)
    except ZeroDivisionError:
        print("Oops! dividing by zero Try again...")
    finally:
        print("finally runs whether there is an exception or no exception")
    print(" ")

# raise example without being handled
print("now we raise an exception without it being handled")
raise ZeroDivisionError
```

Ln: 61 Col: 59

Running of example code