



CS110: Introduction to Computer Science



Dianna Xu

1




The Limits of `int`

- What is 100!?

```
9332621544394415268169923885626670049071596826438162
146859296389521759993229915608941463976156518286253
697920827223758251185210916864000000000000000000000
00
```

2




The Limits of Int

- Newer versions of Python can handle it, but...

```
Python 1.5.2 (#0, Apr 13 1999, 10:51:12) [MSC 32 bit (Intel)] on win32
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
>>> import fact
>>> fact.main()
Please enter a whole number: 13
13
12
11
10
9
8
7
6
5
4
Traceback (innermost last):
  File "<pyshell#1>", line 1, in ?
    fact.main()
  File "C:\PROGRAMS-1\PYTHON-1.2\fact.py", line 5, in main
    fact=fact*factor
OverflowError: integer multiplication
```


3



The Limits of Int

- What's going on?
 - While there are an infinite number of integers, there is a finite range of ints that can be represented.
 - This range depends on the number of *bits* a particular CPU uses to represent an integer value. Typical PCs use 32 bits.

4




Bits

- Bit – binary digit (0 or 1)

Bit 2	Bit 1
0	0
0	1
1	0
1	0


5



The Limits of Int

- Say our PCs use 32 bits
- That means there are 2^{32} possible values, centered at 0.
- This range then is -2^{31} to $2^{31}-1$. We need to subtract one from the top end to account for 0.
- We can test this with an old version of Python.

6




The Limits of Int

```

Python 1.5.2 (#0, Apr 13 1999, 10:51:12) [MSC 32 bit (Intel)] on
win32
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
>>> 2**30
1073741824
>>> 2**31
Traceback (innermost last):
  File "<pyshell#3>", line 1, in ?
    2**31
OverflowError: integer pow()
>>>

```

7



The Limits of Int

- It blows up between 2^{30} and 2^{31} as we expected. Can we calculate $2^{31}-1$?


```

>>> 2**31-1
Traceback (innermost last):
  File "<pyshell#5>", line 1, in ?
    2**31-1
OverflowError: integer pow()

```

- What happened?

8



The Limits of Int


- We need to be more clever!
- $2^{31} = 2 \cdot 2^{30} = 2^{30} + 2^{30}$
- $2^{31}-1 = 2^{30}-1 + 2^{30}$

```

>>> 2**30-1+2**30
2147483647
>>> 2147483647+1
Traceback (innermost last):
  File "<pyshell#7>", line 1, in ?
    2147483647+1
OverflowError: integer addition
>>>

```

9



The Limits of Int


- What have we learned?
 - The largest int value we can represent is 2147483647
- How do modern versions of Python handle this?

```

Python 2.3.3 (#51, Dec 18 2003, 20:22:39) [MSC v.1200 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2**40
1099511627776L

```

10



Handling Large Numbers: floats

- Does switching to *float* data types get us around the limitations of *ints*?
- If we initialize the accumulator to 1.0, we get


```

>>> main()
Please enter a whole number: 15
The factorial of 15 is 1.307674368e+012

```

- We no longer get an exact answer!

11



Handling Large Numbers: floats

- Very large and very small numbers are expressed in *scientific* or *exponential notation*.
- $1.307674368e+012$ means $1.307674368 \cdot 10^{12}$
- Here the decimal needs to be moved right 12 decimal places to get the original number

12

Handling Large Numbers: floats

- Floats are approximations
- Floats allow us to represent a larger range of values, but with lower precision.
- Python has a solution, the `long int`!
- `long ints` (`long`) are not a fixed size and expand to handle whatever value it holds.

13

Handling Large Numbers: Long ints

- To get a long int, put “L” on the end of a numeric literal.
- 5 is an `int` representation of five
- 5L is a `long` representation of five
- Calculations involving `long` produce `long` results.
- Newer versions of Python automatically convert your `ints` to `longs` when they grow so large as to overflow.

14

Type Conversions

- We know that combining an `int` with an `int` produces an `int`, and combining a `float` with a `float` produces a `float`.
- What happens when you mix an `int` and `float` in an expression?
`x = 5.0 / 2`
- What do you think should happen?

15

Type Conversions

- For Python to evaluate this expression, it must either convert 5.0 to 5 and do an integer division, or convert 2 to 2.0 and do a floating point division.
- Converting a `float` to an `int` will lose information
- `int` literals can be converted to floats by adding “.0”

16

Type Conversion


- In *mixed-typed expressions* Python will convert `ints` to `floats`.
- Similarly Python will convert `ints` to `longs`

17

Explicit Type Conversion

- Sometimes we want to control the type conversion. This is called *explicit typing*.
- `average = sum / n`
- If the numbers to be averaged are 4, 5, 6, 7, then `sum` is 22 and `n` is 4, so `sum/n` is 5, not 5.5!


18



Explicit Type Conversions

- To fix this problem, tell Python to change one of the values to floating point:
`average = float(sum)/n`
- We only need to convert the numerator because now Python will automatically convert the denominator.


19



Type Conversions

- Why doesn't this work?
`average = float(sum/n)`
- Python also provides `int()` and `long()` functions to convert numbers into `ints` and `longs`.
- Converting to integers might truncate!

20



Type Conversions

- The `round` function returns a float, rounded to the nearest whole number.
- .5 rounds up

21