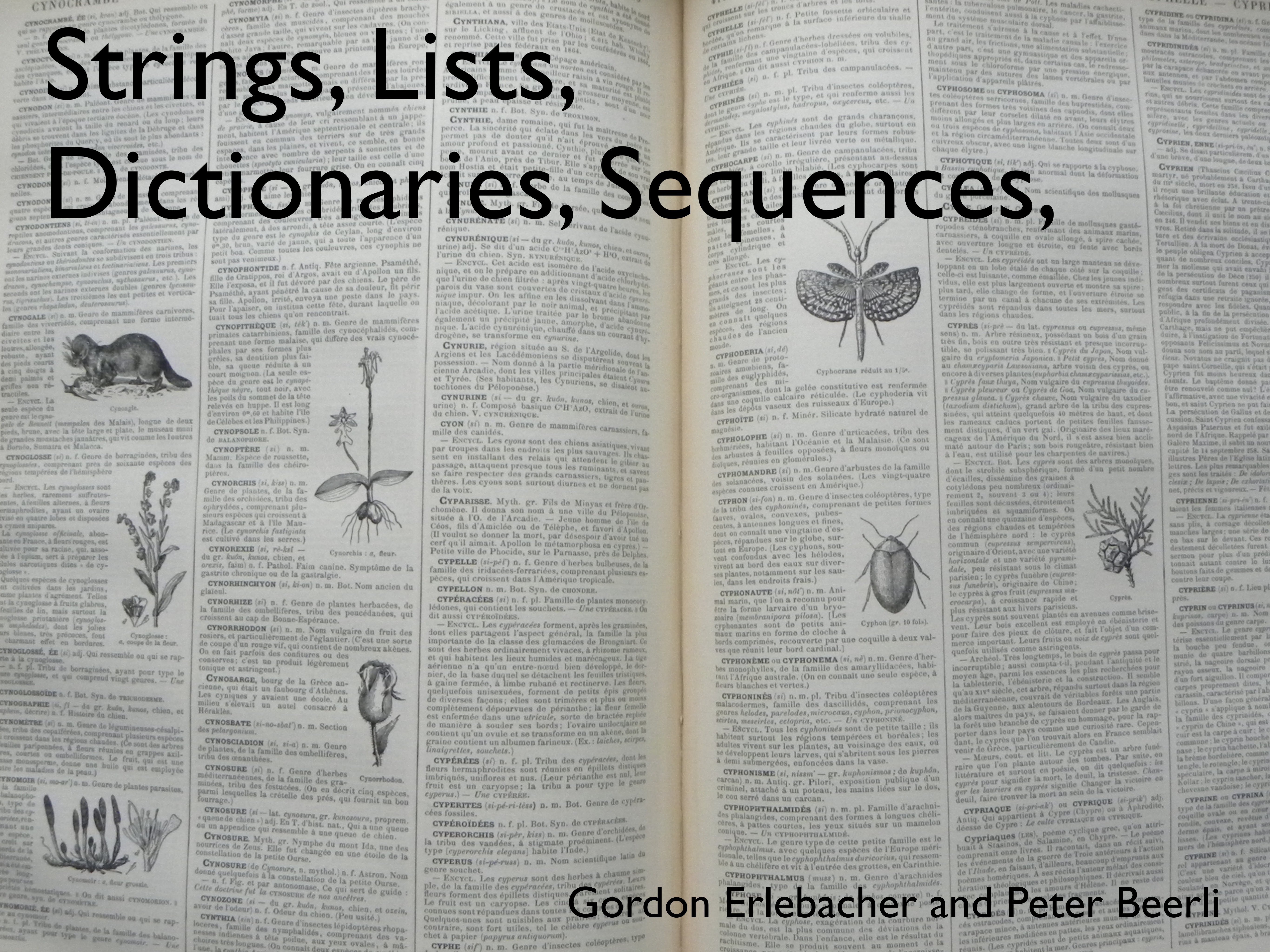


Strings, Lists, Dictionaries, Sequences,

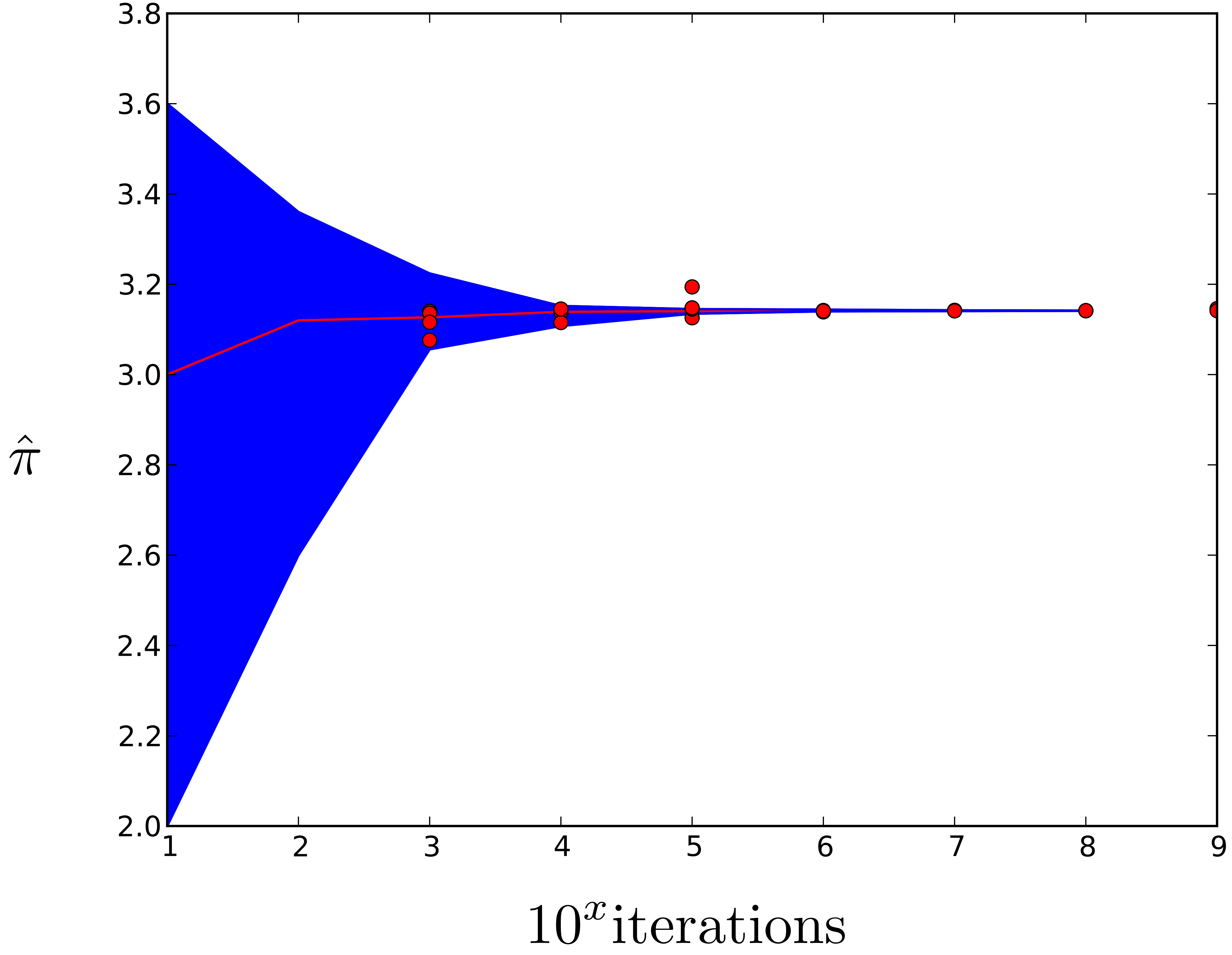
Gordon Erlebacher and Peter Beerli

Python programming steps

Enter in file:

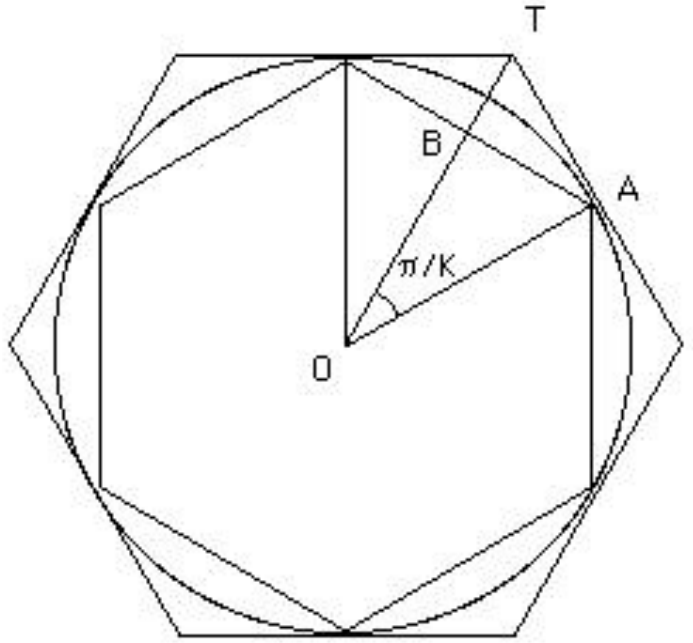
```
#!/usr/bin/env python
from __future__ import print_function
import random
import math
#initialize variables
i = 0
n = 100000
r = 1.0
circle = 0.0
square = 0.0
# Do many times:
while i < n:
     i = i + 1
     #    draw x, y coordinate
     x = random.uniform(0.0,r)
     y = random.uniform(0.0,r)
     #    calculate d from center
     d = math.sqrt(x**2 + y**2)
     #    check whether d < r:
     if d < r:
        #       True: add 1 to circle
        circle = circle + 1
        #       False: do nothing

     #    add 1 to square
     square = square + 1
# print pi: ratio circle/square * 4
print (“pi = ” + str(circle/square * 4.0))
```

Our Pi estimates



History of π

Archimedes (300 BC)
using 96-side polygons



$$223/71 < \pi < 22/7$$

3.140845070422535 3.142857142857142 8

Ptolemy	(c. 150 AD)	3.1416
Zu Chongzhi	(430-501 AD)	$^{355}/_{113}$
al-Khwarizmi	(c. 800)	3.1416
al-Kashi	(c. 1430)	14 places
Viète	(1540-1603)	9 places
Roomen	(1561-1615)	17 places
Van Ceulen	(c. 1600)	35 places

James Gregory
1638-1675

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots$$

History of π

1699: Sharp used Gregory's result to get 71 correct digits
1701: Machin used an improvement to get 100 digits and the following used his methods:
1719: de Lagny found 112 correct digits
1789: Vega got 126 places and in 1794 got 136
1841: Rutherford calculated 152 digits and in 1853 got 440
1873: Shanks calculated 707 places of which 527 were correct

Very soon after Shanks' calculation a curious statistical freak was noticed by De Morgan, who found that in the last of 707 digits there was a suspicious shortage of 7's. He mentions this in his Budget of Paradoxes of 1872 and a curiosity it remained until 1945 when Ferguson discovered that Shanks had made an error in the
528th place, after which all his digits were wrong. In 1949 a computer was used to calculate π to 2000 places. In this and all subsequent computer expansions the number of 7's does not differ significantly from its expectation, and indeed the sequence of digits has so far passed all statistical tests for randomness.

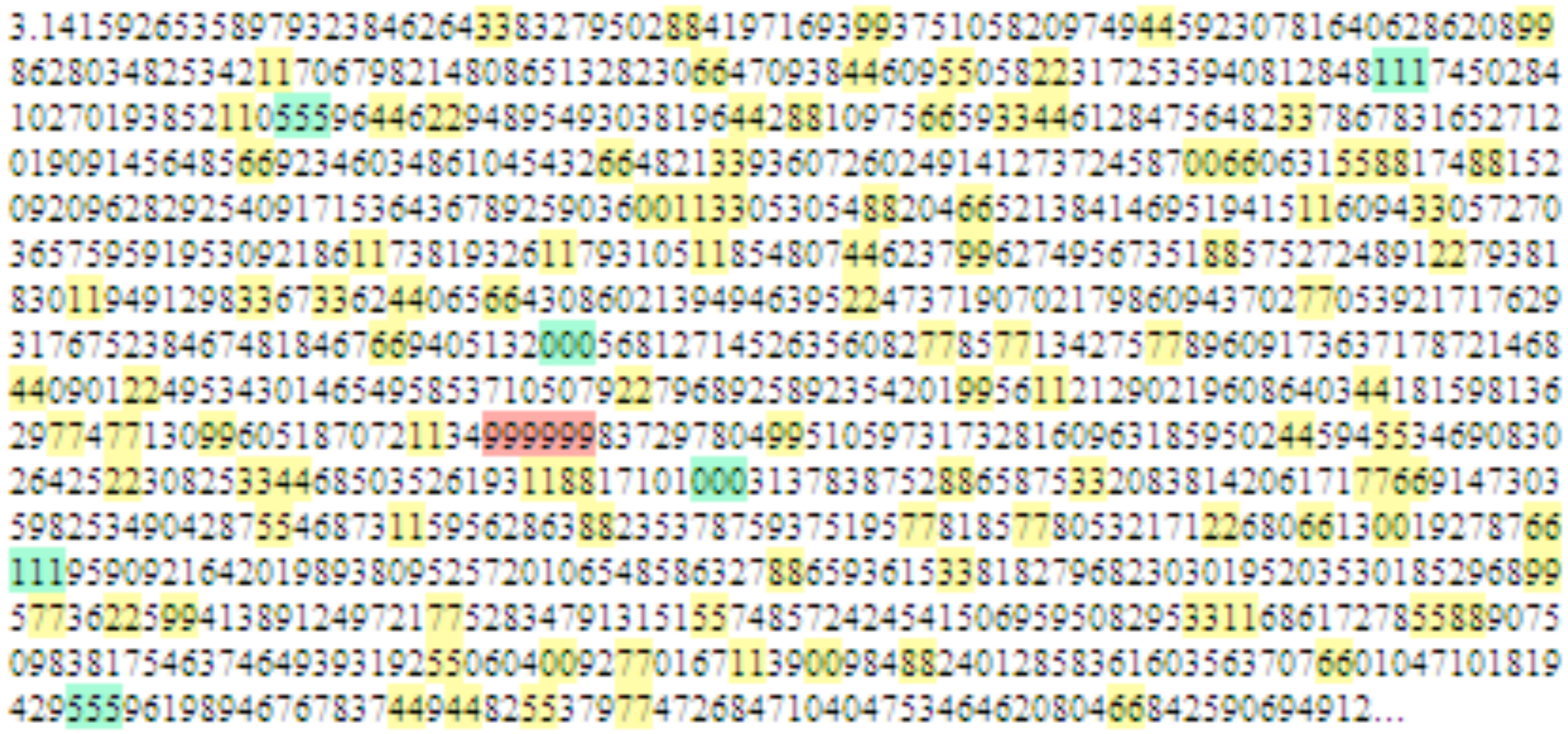
Buffon's needle experiment. If we have a uniform grid of parallel lines, unit distance apart and if we drop a needle of length $k < 1$ on the grid, the probability that the needle falls across a line is $2k/\pi$. Various people have tried to calculate π by throwing needles. The most remarkable result was that of Lazzerini (1901), who made 34080 tosses and got
$\pi = 355/113 = 3.1415929$
which, incidentally, is the value found by Zu Chongzhi. This outcome is suspiciously good, and the game is given away by the strange number 34080 of tosses. Kendall and Moran comment that a good value can be obtained by stopping the experiment at an optimal moment. If you set in advance how many throws there are to be then this is a very inaccurate way of computing π. Kendall and Moran comment that you would do better to cut out a large circle of wood and use a tape measure to find its circumference and diameter.

π

In the State of Indiana in 1897 the House of Representatives unanimously passed a Bill introducing a new mathematical truth:

Be it enacted by the General Assembly of the State of Indiana: It has been found that a circular area is to the square on a line equal to the quadrant of the circumference, as the area of an equilateral rectangle is to the square of one side. (Section I, House Bill No. 246, 1897)

The Senate of Indiana showed a little more sense and postponed indefinitely the adoption of the Act!



```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899
8628034825342117067982148086513282306647093844609550582231725359408128481117450284
1027019385211055596446229489549303819644288109756659334461284756482337867831652712
0190914564856692346034861045432664821339360726024914127372458700660631558817488152
0920962829254091715364367892590360011330530548820466521384146951941511609433057270
3657595919530921861173819326117931051185480744623799627495673518857527248912279381
8301194912983367336244065664308602139494639522473719070217986094370277053921717629
3176752384674818467669405132000568127145263560827785771342757789609173637178721468
4409012249534301465495853710507922796892589235420199561121290219608640344181598136
2977477130996051870721134999999837297804995105973173281609631859502445945534690830
2642522308253344685035261931188171010003137838752886587533208381420617177669147303
5982534904287554687311595628638823537875937519577818577805321712268066130019278766
1119590921642019893809525720106548586327886593615338182796823030195203530185296899
5773622599413891249721775283479131515574857242454150695950829533116861727855889075
0983817546374649393192550604009277016711390098488240128583616035637076601047101819
429555961989467678374494482553797747268471040475346462080466842590694912...
```

Strings

```
>>> a = "going to class“
>>> b = 'going to class‘
>>> c = """going to class"""“
>>> d = r'going to class'
```

‘\n’ is a carriage return

raw format

using **Triple quotes**

```
In [39]: a="""going
   ....:     to school
   ....:   late
   ....: """

In [40]: a
Out[40]: 'going \n    to school\n  late\n'

In [41]: print a
going
    to school
  late
```

Advantage of raw format

- I want to encode the string: “path\to\file”
- “\” is a special character and one must do: “\\”

```
In [72]: a='\a\b\c'

In [73]: a
Out[73]: '\x07\x08\\c'

In [74]: print a
\c

In [75]: a="""\a\b\c"""

In [76]: print a
\c
```

```
In [77]: a=r'\a\b\c'

In [78]: print a
\a\b\c

In [79]: a='\\a\\b\\c'

In [80]: print a
\a\b\c
```

Booleans

- A boolean is either True or False
- In some languages, -1 if true, all else is false, in others, 0 is false, all else is true
- In Python : the number 0 is false, all other numbers are true. Do not assume this!!!
  - None, (), [], 0, “” returns false
    - bool(None) ==> False
    - bool([]) ==> False
    - bool(34) ==> True
    - bool(None or 34) ==> True
    - bool(34 and (not 0 or “”)) ==> True

What is None?

```
In [1]: bool(trip)
---------------------------------------------------------------------------
NameError                                 Traceback (most recent
call last)
<ipython-input-1-2b29e319ed42> in <module>()
----> 1 bool(trip)

NameError: name 'trip' is not defined

In [2]: trip=None
In [3]: bool(trip)
Out[3]: False

In [4]: trip=""
In [5]: bool(trip)
Out[5]: False

In [6]: trip=3
In [7]: bool(trip)
Out[7]: True
```

- **None** is the absence of definition
- **""** is the empty string

Lists

- A collection of objects
- There is an order
  - a[0] comes before a[1]
- List elements can be modified (mutable)
- Heterogeneous (strings, ints, floats, functions)

Lists

- a = []   #empty list
- a.extend([3,4])   # a = [3,4]
- a.append([3,4])   # a = [3,4,[3,4]]   (add single element)
- a.extend([3,4])   # a = [3,4,[3,4],3,4] (add element**s**)
- a[1] = ‘pyth‘   # a = [3,‘pyth’,[3,4]]
- mutable
- heterogeneous
- type(a)   # <type ‘list’>

List Initialization

- Use the “*” operator
  - a = [3] * 10   ==> [3,3,....,3]
  - b = [3,4,5] * 7 ==> [3,4,5,3,4,5,...,3,4,5]
  - c = ‘hu’ * 5 ==> ‘huhuhuhuhu’
- Use an iterator
  - a = list(xrange(5)) # [0,1,2,3,4]
  - print xrange(5)   # xrange(5)
  - type(xrange(5))   # <type ‘xrange’> (iterator)

```
class xrange(object)
 |  xrange(stop) -> xrange object
 |  xrange(start, stop[, step]) -> xrange object
 |
 |  Like range(), but instead of returning a list, returns an object that
 |  generates the numbers in the range on demand.  For looping, this is
 |  slightly faster than range() and more memory efficient.
```

xrange() obsolete in Python 3
it got rerplaced by range()

Special lists

- range(5)   # returns 0,1,2,3,4
- xrange(5)   # iterator object
- dir([])
  ['**__add__**', '__class__', '__contains__', '__delattr__', '__delitem__', '__delslice__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '**__getslice__**', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '**__mul__**', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__setslice__', '__sizeof__', '__str__', '__subclasshook__', '**append**', '**count**', '**extend**', 'index', '**insert**', 'pop', 'remove', '**reverse**', '**sort**']
- a = range(5)
  a.reverse().sort()   # in place reversion followed by a sort

References

```
>>> a = range(8)
>>> a
[0, 1, 2, 3, 4, 5, 6, 7]
>>> b = a
>>> b[5] = 'class'
>>> a
[0, 1, 2, 3, 4, 'class', 6, 7]
>>>
```

b is a reference to **a**

any change to an element to **b** also changes **a**

Slicing

```
>>> a = range(8)
>>> a
[0, 1, 2, 3, 4, 5, 6, 7]
>>> b = a[:]
>>> b[5] = 'class'
>>> a
[0, 1, 2, 3, 4, 5, 6, 7]
>>> b
[0, 1, 2, 3, 4, 'class', 6, 7]
```

b is a copy of **a**
Changing an element of **b**
does *not* change **a**

a[:] is an example of a splice
Slices are **copies** of a *subset* of the original array

Slicing

- a = range(5) # [0,1,2,3,4]
- a[3:5]   # [3,4]
- a[3:]   # [3,4]
- a[-3]   # 2
- a[-1]   # 4
- a[-3:-1]   # [2,3]

List Errors

- c[3] = 4
  - # Name error: c not defined
- c = []
- c[3] = 2
  - # Index error: list assignment index out of range

Sequences, Tuple

- A sequence is similar to a list, *except* that it cannot be modified
  - immutable

Sequence (Tuple)

- Immutable (= Cannot be changed)
- a = (2,3,5)
- a[1] = 3   # exception
- a = (3)   # not a sequence
- a = 3,   # or (3,) is a sequence

Sequence

Immutable object

```
>>> a = (1,2,3)
>>> type(a)
<type 'tuple'>
>>> dir(a)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
'__eq__','__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__','__getslice__', '__gt__',
'__hash__', '__init__', '__iter__', '__le__', '__len__','__lt__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'count', 'index']
```

```
>>> a = ('notes',3,-34,7)
>>> a[1] = 3                    # immutable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> a + (5,6,7)                 # a.__add__((5,6,7))
('notes', 3, -34, 7, 5, 6, 7)
>>> len(a)                      # a.__len__
4
>>> a*3                         #  a.__mul__(3)
('notes', 3, -34, 7, 'notes', 3, -34, 7, 'notes', 3, -34, 7)
>>> a[5]                        #  a.__getitem__(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: tuple index out of range
>>> a[2]
-34
>>> a
('notes', 3, -34, 7)
```

Dictionaries

- A list can contain anything, but there is an order: the list can be indexed.
- A dictionary (also called *hash*) is a collection of (key:value) pairs
- There is no indexing
- The key can be any immutable object
  - int, float, long, sequence, string
- The value can be any object (mutable, immutable)
  - list, class, function, etc.

Dictionary

- a= {}   # or a = dict() (not common)
- c = {‘1’:‘peter’, 2 :‘lucrezia’}
  - c[‘1’] #   ‘peter’
  - c[1]   # error (key not defined)
- a[3] = ‘gordon’
- a[‘egg’] = ‘steamed’
- a[(3,4,5)] = [‘class’, [3,4,5], 6]
- b = a[(3,4,5)][2]   returns 6
- dict[**key**] = **value**

Dictionary

- variable[key] = value
- key can be:
  - any immutable object
  - string, int, float, sequence

Dictionary

- dir({})
- ['__class__', '__cmp__', '**__contains__**', '__delattr__', '**__delitem__**', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '**__getitem__**', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '**__len__**', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', '**has_key**', '**items**', '**iteritems**', '**iterkeys**', '**itervalues**', '**keys**', 'pop', 'popitem', 'setdefault', 'update', 'values']

```
>>> a = {}
>>> a[3] = 'gor'
>>> a['frank'] = 'code'
>>> a['grow'] = 35.5
>>> a.keys()
['frank', 3, 'grow']
>>> del a['grow']
>>> a.keys()
['frank', 3]
>>> len(a)
2
>>> c = a[‘grow’]
>>> c = a.__getitem__('frank')
>>> c
‘code’
```

Dictionary Members

```
>>> a = {}
>>> a[3] = (3,5,6)
>>> a[('joe', 34)] = 'exam'
>>> a['area'] = 3.56
>>> a.keys()
[3, ('joe', 34), 'area']
>>> a.values()
[(3, 5, 6), 'exam', 3.5600000000000001]
>>> a.itervalues()
<dictionary-valueiterator object at 0x374a80>
>>> a.has_key((3,5))
False
>>> a.has_key((3,5,6))
False
>>> a.has_key(('joe',34))
True
>>>
```

```
>>> a={}
>>> a[1]='peter'
>>> a[2]='jasmin'


>>> a.items()
[(1, 'peter'), (2, 'jasmin')]


>>> a.keys()
[1, 2]
>>> a.values()
['peter', 'jasmin']
```

Sets

- A set is a collection of objects
- There is no order to these objects
- Each element in a set is unique
  - contrary to a list
  - a list a = [1,2,3,3,3]   can contain the integer 3 three times
  - s = set((1,2,3,3,3)) or set([1,2,3,3,3]) returns set([1,2,3]) (the other two 3's are removed)

Set

- a = set()
- dir(a)
- ['__and__', '__class__', '__cmp__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__iand__', '__init__', '__ior__', '__isub__', '__iter__', '__ixor__', '__le__', '**__len__**', '__lt__', '__ne__', '__new__', '__or__', '__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__xor__', 'add', 'clear', 'copy', '**difference**', '**difference_update**', 'discard', '**intersection**', '**intersection_update**', '**isdisjoint**', '**issubset**', '**issuperset**', 'pop', 'remove',

Unique words with Sets

- Assume the following task:
  - given two books, what are the words common to both (plurals and other inflections count as separate words)
- Solution
  - collect all the words from book *A* into setA, and collect all the words from book *B* into setB. The required set of unique words is then simply
  - unique_words = setA.intersect(setB)