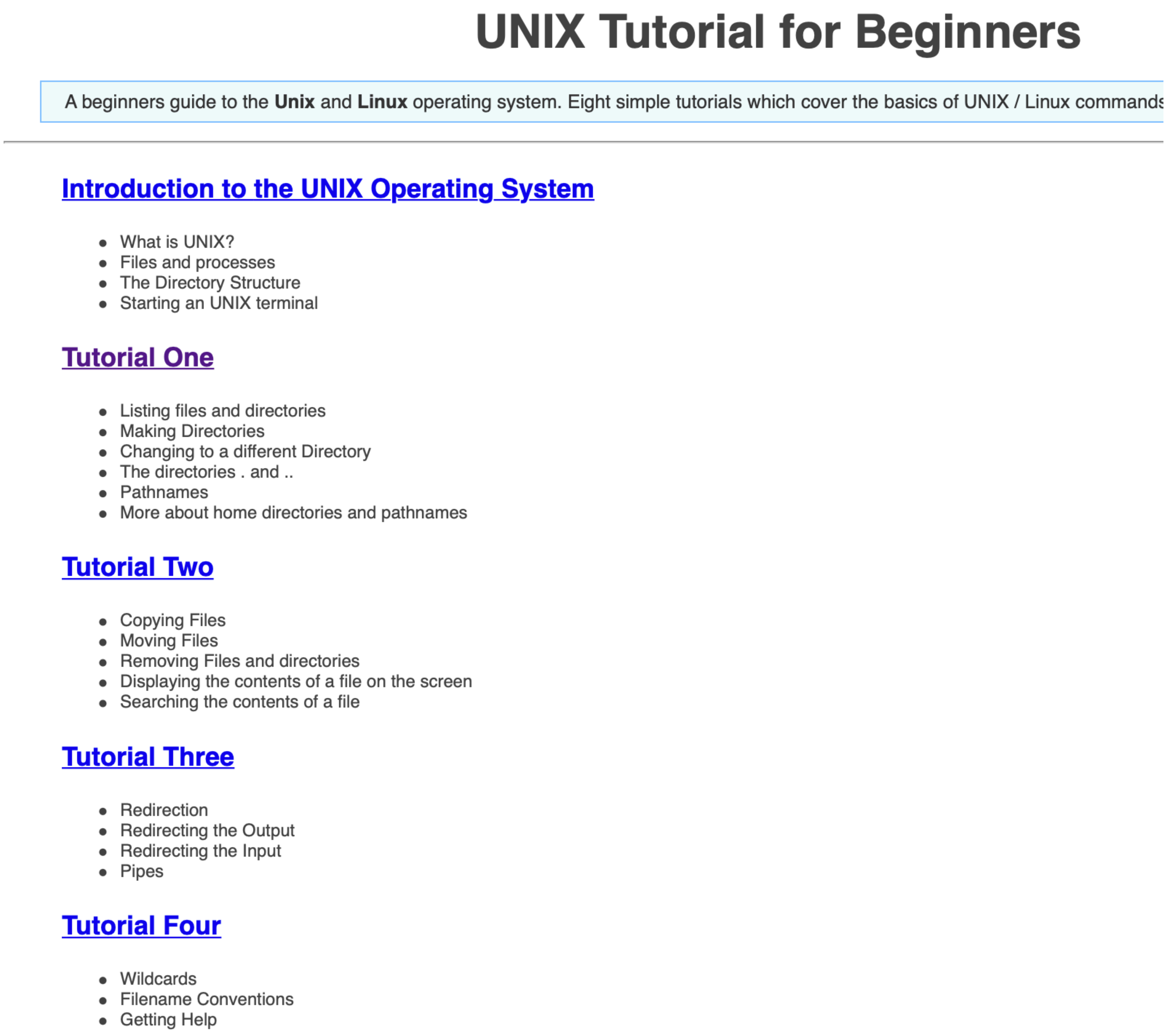
ISC-4304: Python baby steps

http://www.ee.surrey.ac.uk/Teaching/Unix/



UNIX Tutorial for Beginners

A beginners guide to the **Unix** and **Linux** operating system. Eight simple tutorials which cover the basics of UNIX / Linux commands

What is programming?

- What is a computer program?
  - "A set of coded **instructions** that enables a machine, especially a computer, to perform a desired **sequence** of operations." – American Heritage Dictionary
- Programming instructions are written using a "programming language"
  - Examples: C/C++, Java, Assembly, Fortran, Cobol, BASIC
  - LOTS of programming languages, different uses for different languages

C++

Back to index

```
// Hello World in C++ (pre-ISO)

#include <iostream.h>

main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Haskell

Back to index

```
-- Hello World in Haskell

main = putStrLn "Hello World"
```

Lisp

Back to index

```
;;; Hello World in Common Lisp

(defun helloworld ()
  (print "Hello World!")
)
```

Assembler-Linux

Back to index

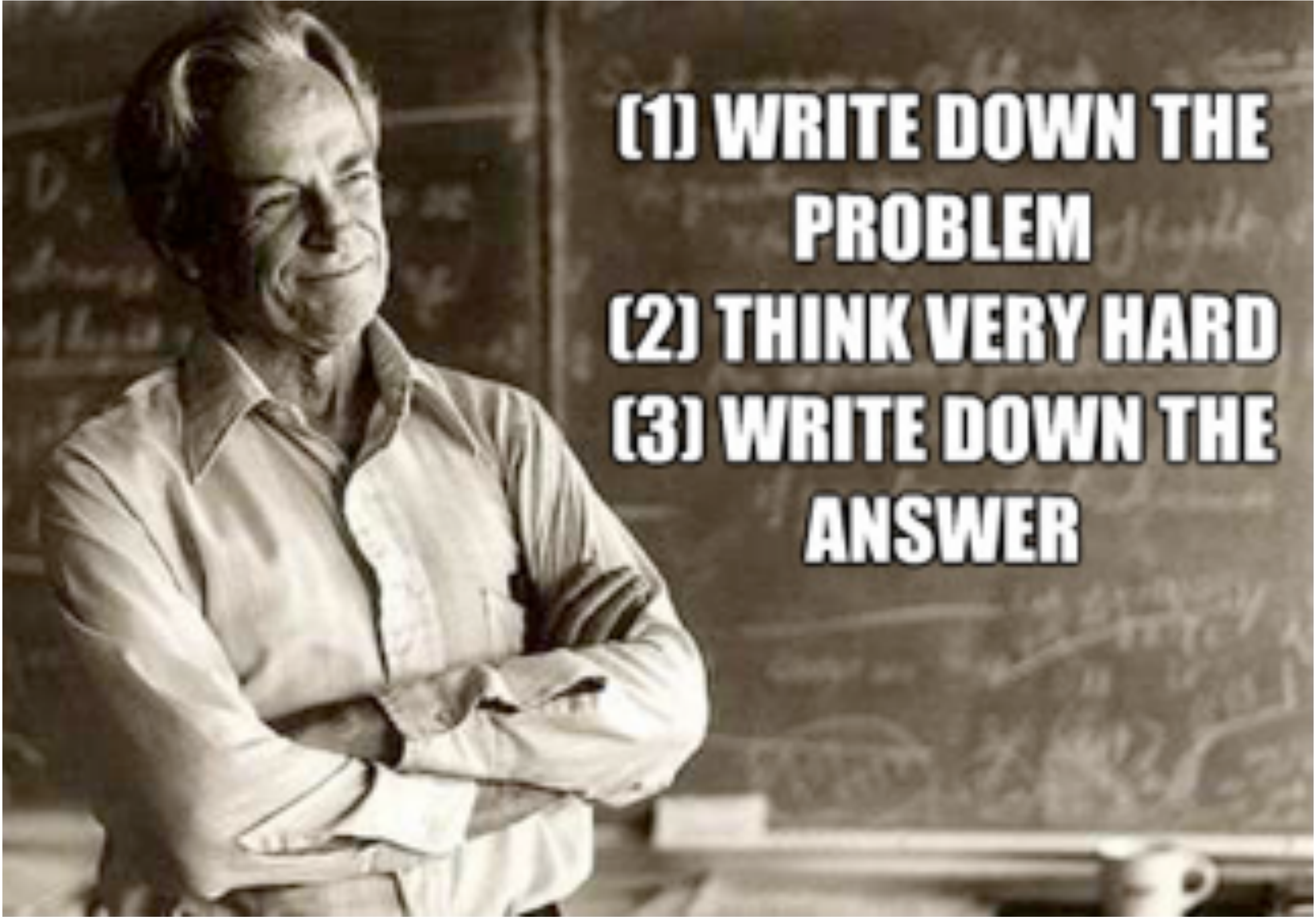
```
    ;; Hello World for the nasm Assembler (Linux)

    SECTION .data

    msg db   "Hello, world!",0xa ;
    len equ     $ - msg

    SECTION .text
    global main

main:
       mov     eax,4       ; write system call
       mov     ebx,1           ; file (stdou)
       mov     ecx,msg         ; string
       mov     edx,len         ; strlen
    int     0x80       ; call kernel

    mov  eax,1         ; exit system call
       mov     ebx,0
       int     0x80        ; call kernel
```

http://helloworldcollection.de

Hello World has been implemented in just about every programming language on the planet. This collection includes **585 Hello World programs** in as many more-or-less well known programming languages, plus **78 human languages**.

Terminology

- **Computer program** – a set of instructions that tell a computer exactly what to do
  - The instructions might tell the computer to add up a set of numbers, or compare two numbers and make a decision based on the result, or whatever.
- **Programming language** – a language used by humans to program computers
  - e.g., Fortran, Cobol, Basic, Pascal, C, C++, Java, Perl
- **Compiler** – translates a computer program written in a human-readable computer language (like C++) into a form that a computer can execute
  - You have probably seen .exe files or .app ‘files’ on your computer.
  - These executable files are the output of compilers.
  - They contain executables -- machine-readable programs translated from human-readable programs.


(1) WRITE DOWN THE PROBLEM
(2) THINK VERY HARD
(3) WRITE DOWN THE ANSWER

Programming

- Problem solving
  - Logical/methodical way of solving a problem
- Algorithm/abstraction
  - An algorithm is a series of step-by-step instructions that produces a solution to a problem
- Step wise refinement
  - Incrementally adding functionality to a program

Five steps to writing a program

- Define the problem
- Plan the solution
  - pseudocode
- Code the program
  - Using a programming language
- Test and debug
  - Using a compiler
- Document

Python vs C++

- Python is interpreted
- C++ is compiled

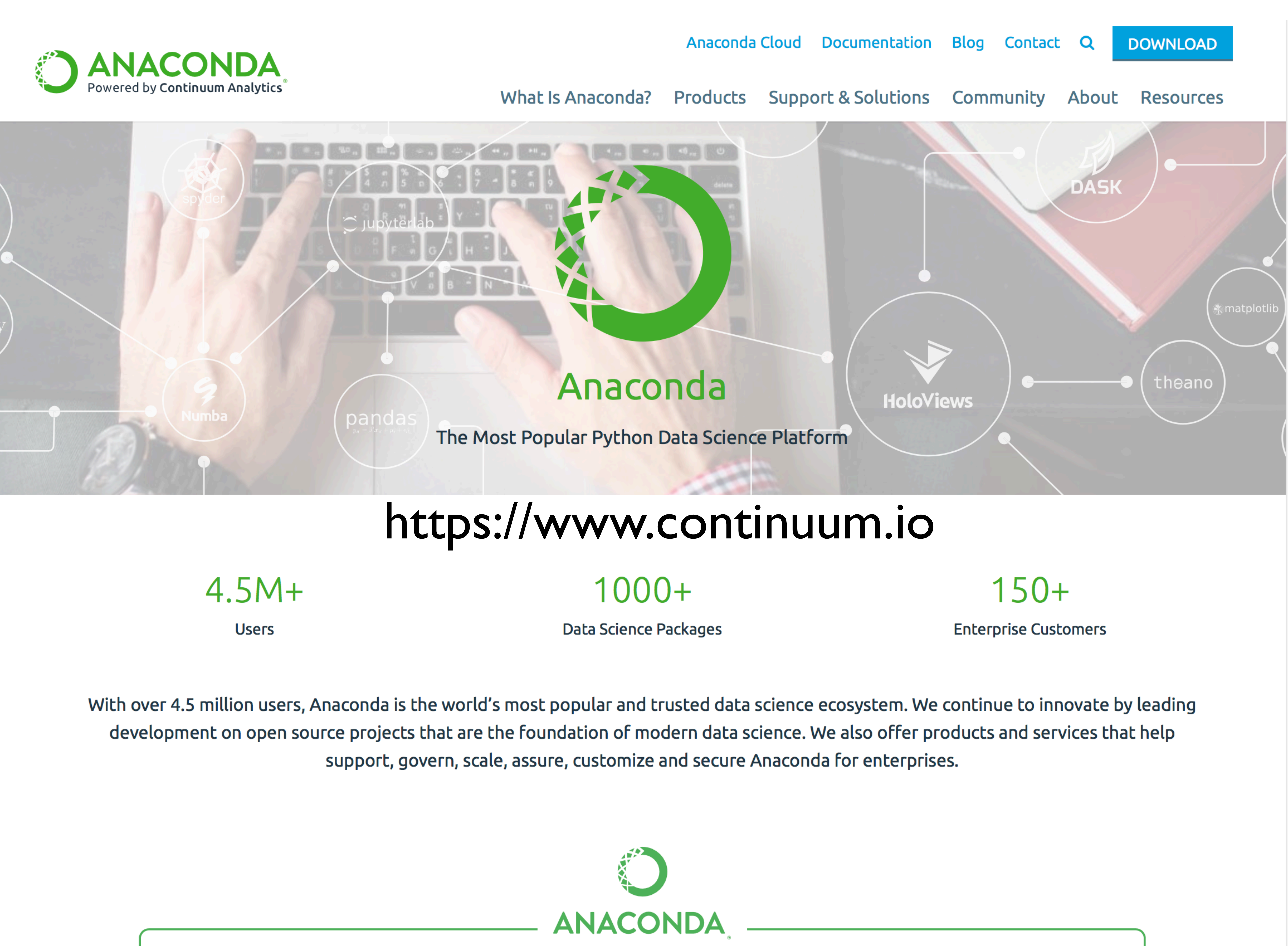
Compiler

A **compiler** is computer software that transforms computer code written in one programming language (the source language) into another programming language (the target language). Compilers are a type of translator that support digital devices, primarily computers. The name *compiler* is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language, object code, or machine code) to create an executable program.[1]

Interpreter

In computer science, an **interpreter** is a computer program that directly executes, i.e. *performs*, instructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program. An interpreter generally uses one of the following strategies for program execution:

- parse the source code and perform its behavior directly;
- translate source code into some efficient intermediate representation and immediately execute this;
- explicitly execute stored precompiled code[1] made by a compiler which is part of the interpreter system.



Anaconda

The Most Popular Python Data Science Platform

https://www.continuum.io

4.5M+	1000+	150+
Users	Data Science Packages	Enterprise Customers

With over 4.5 million users, Anaconda is the world's most popular and trusted data science ecosystem. We continue to innovate by leading development on open source projects that are the foundation of modern data science. We also offer products and services that help support, govern, scale, assure, customize and secure Anaconda for enterprises.

What is Python?

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants, on the Mac, and on PCs under MS-DOS, Windows, Windows NT, and OS/2.

To find out more, start with *The Python Tutorial*. The Beginner's Guide to Python links to other introductory tutorials and resources for learning Python.

http://docs.python.org/2/faq/general.html#id1

Why is it called Python

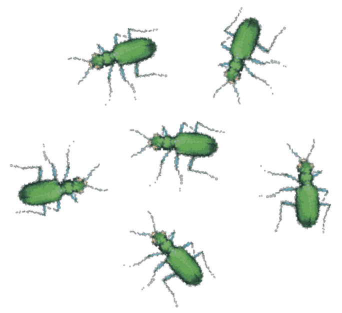
When he began implementing Python, Guido van Rossum was also reading the published scripts from "Monty Python's Flying Circus", a BBC comedy series from the 1970s. Van Rossum thought he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python.

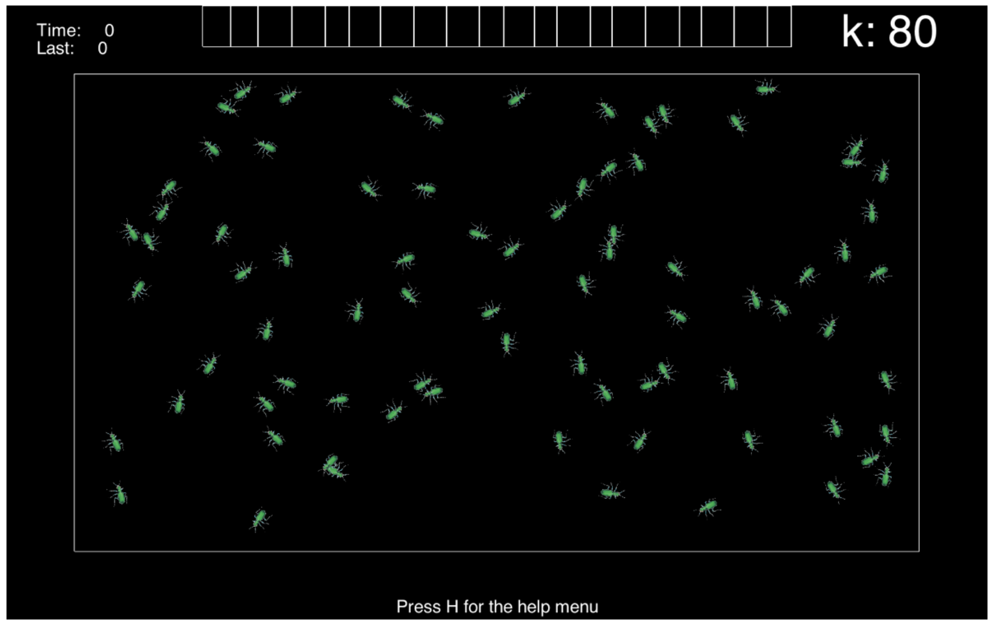
"The most important thing in the programming language is the name. A language will not succeed without a good name. I have recently invented a very good name and now I am looking for a suitable language." — Donald Knuth

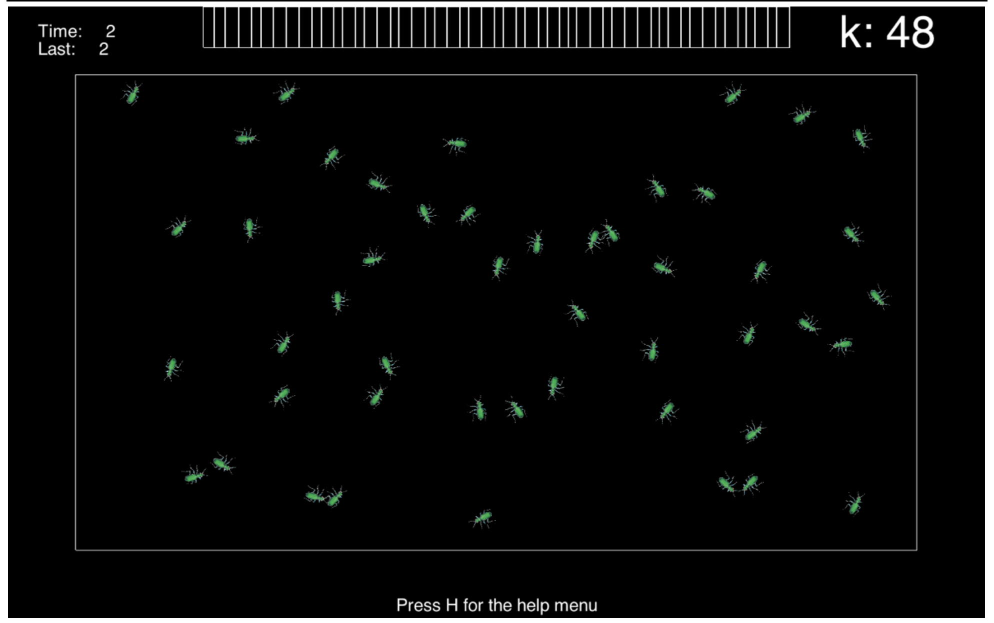
http://docs.python.org/2/faq/general.html#id1

https://www.slideshare.net/SidharthNadhan/learn-python-in-20-minutes

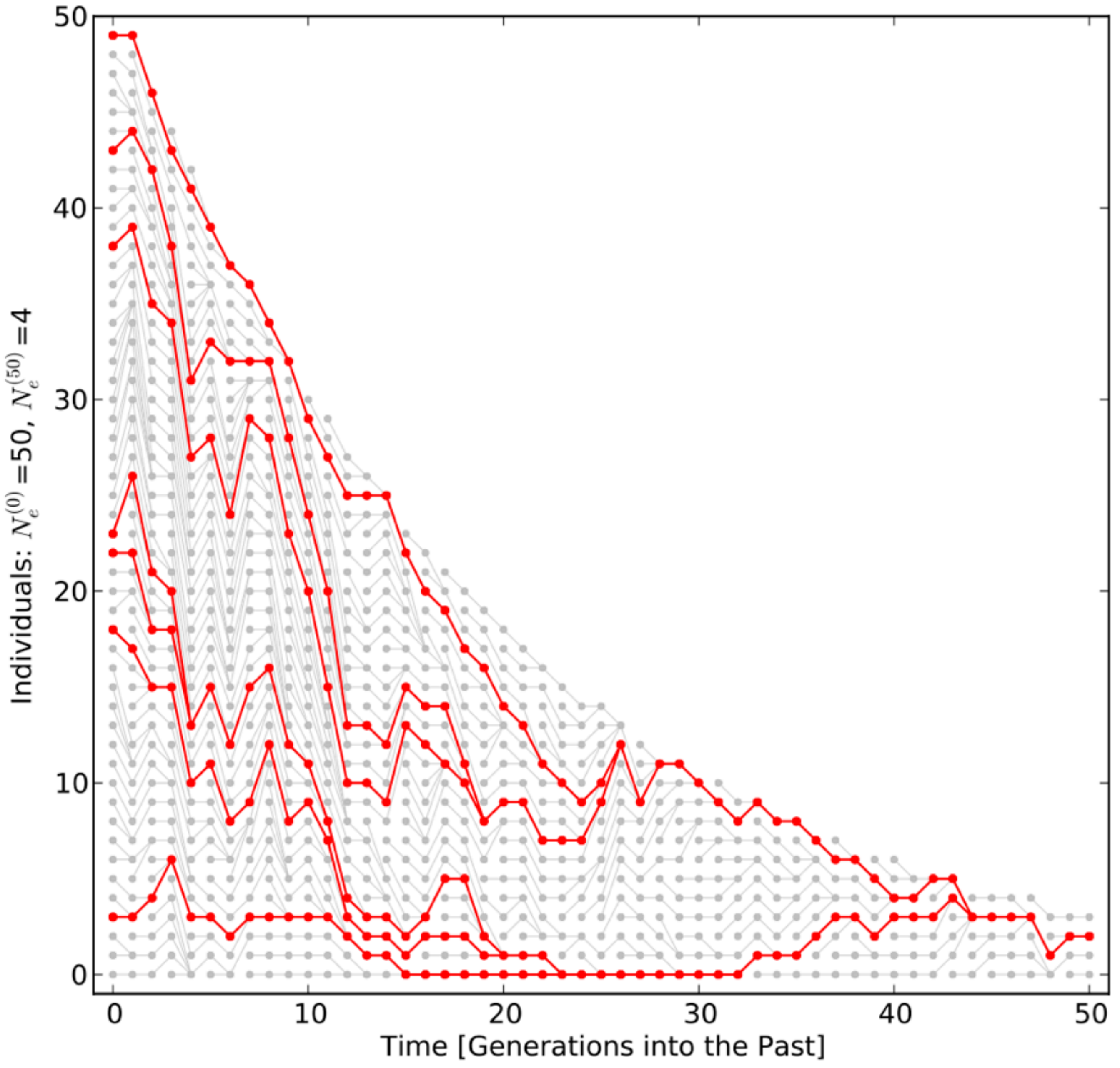
Examples

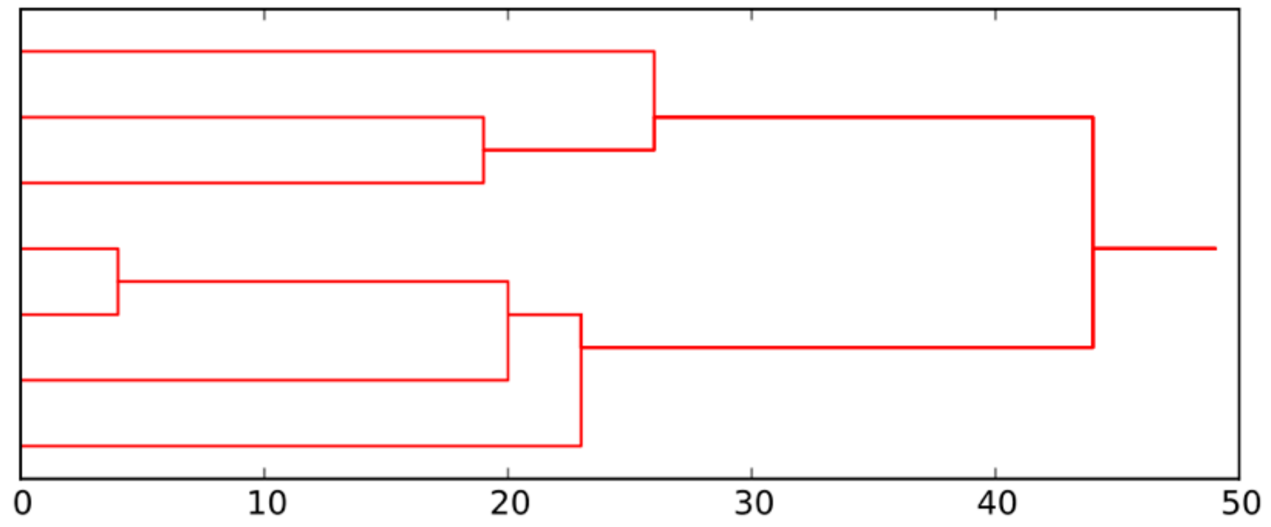

bugs in a box





popvizard





Python baby steps: Python as a calculator

Python baby steps: we learn how to calculate Pi

Open two terminal windows that point to the same directory.
Use the text editor

nano or gedit (or vi or emacs [for geeks])

to edit a file in one window and in the other execute that file with something like this

```
python file
```

for python programs I often use the .py extension, for our examples use hello.py and pi.py as file names.
Again make sure that both terminal window point to the same directory (use pwd to check).

Python programming steps

my first program

Enter in file:

```
print “Hello world”
```

Result:

```
Hello world
```

Python programming steps

printing to screen

Enter in file:

```
a = 1
b = 2
print(a, b)
print ("—")
print(a)
print (b)
```

Result:

```
1 2
---
1
1
```

Python programming steps

Looping

Enter in file:

```
a = 0          # the # is a comment, a is assigned zero
b = 10         # b is assigned 10
while a < b:   # while a is smaller than b do the following
  a = a + 1    # add 1 to a and assign the result to a
  print a,     # print a, the ‘,’says add a blank
               # the indentation is important in python because
               # it marks that all the material belongs to the
               # while statement, a “:” marks such a statement.
```

Result:

```
 1 2 3 4 5 6 7 8 9 10
```

Python programming steps

Looping

Enter in file:

```
a = range(10)   # creates a list from 0 to 9
b = range(1,11) # creates a list from 1 to 10
# loop over all b and print a running sum of the square of b[i]
sum = 0
for bi in b:
  sum = sum + bi * bi
  print sum,
print
```

Result:

```
  1 5 14 30 55 91 140 204 285 385
```

Python programming steps

decisions

Enter in file:

```
a = 0
b = 10
c = 5
while a < b:  # loop as long a is smaller than b
  a = a + 1   # increase a
  if a < c:   # if a is smaller than c
     print a, #Python3: print(a,end=‘ ‘) # print a
  else:       # otherwise
     print a*a, #Python3: print(a*a,end=‘ ‘) # print square of a
              #
print “done”  #Python3: print(“done”)  #
```

Result:

```
1 2 3 4 25 36 49 64 81 100 done
```

Python programming steps

list comprehension

Enter in file:

```
a = []                  # a is initialized as an empty list
print “a=”,a
b = [1,2,3,4,”5”]       # b is a list with mixed types
print b
c = b                   # c is a clone of b
b[0] = 5                # changing the first element of b
b=[-1] = “five”            # changing the last element of b
print “b=”,b            # print b
print “c=”,c            # c is just another name for b
c = b[:]                # c now is a indpendent copy of b
b[1] = 21
print “b”,b
print “c=”,c
```

Result:

```
a= []
b= [1, 2, 3, 4, '5']
b= [5, 2, 3, 4, 'five']
c= [5, 2, 3, 4, 'five']
b= [5, 21, 3, 4, 'five']
c= [5, 2, 3, 4, 'five']
```

Python programming steps

list comprehension

Enter in file:

```
a=[0,1,2,3,4,5,6,7,8,9]
b=a[0]
c=a[-1]
d=a[2:4]
e=a[3:-2]
print b
print c
print d
print e
```

Result:

```
0
9
[2, 3]
[3, 4, 5, 6, 7]
```

Python programming steps

strings are funny lists

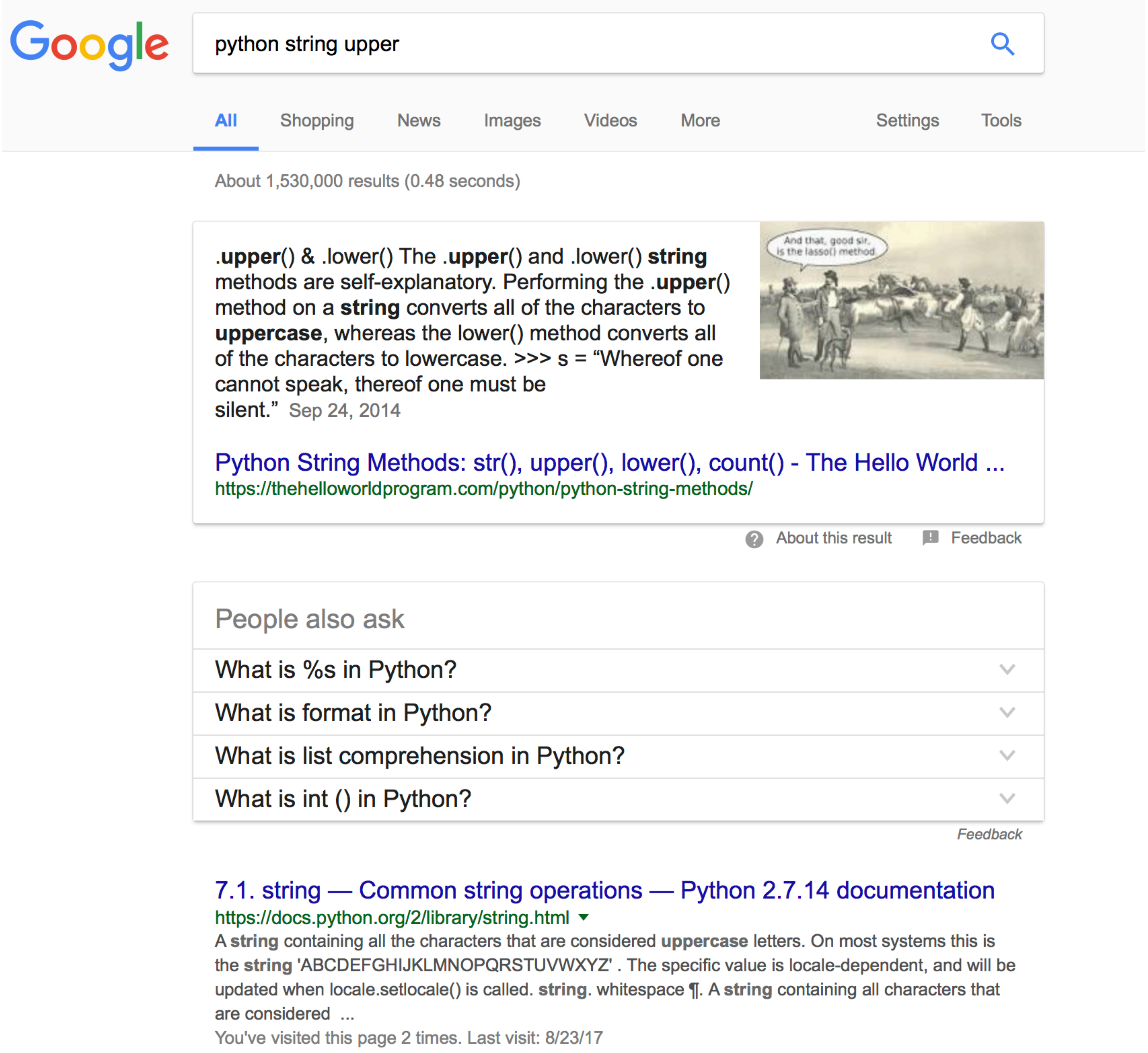
Enter in file:

```
a="the quick fox jumps over the lazy dog"
b=a[-1]
e=a[3:-8]
f=list(a)
g = a.split()
h = a.upper()
print b
print e
print f
print g
print h
```

Result:

```
g
 quick fox jumps over the
['t', 'h', 'e', ' ', 'q', 'u', 'i', 'c', 'k', ' ', 'f', 'o', 'x', ' ', 'j',
'u', 'm', 'p', 's', ' ', 'o', 'v', 'e', 'r', ' ', 't', 'h', 'e', ' ', 'l',
'a', 'z', 'y', ' ', 'd', 'o', 'g']
['the', 'quick', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
THE QUICK FOX JUMPS OVER THE LAZY DOG
```

google
python string upper



python string upper

About 1,530,000 results (0.48 seconds)

.**upper**() & .lower() The .**upper**() and .lower() **string** methods are self-explanatory. Performing the .**upper**() method on a **string** converts all of the characters to **uppercase**, whereas the lower() method converts all of the characters to lowercase. >>> s = “Whereof one cannot speak, thereof one must be silent.” Sep 24, 2014

Python String Methods: str(), upper(), lower(), count() - The Hello World ...
https://thehelloworldprogram.com/python/python-string-methods/

About this result   Feedback

People also ask

What is %s in Python?

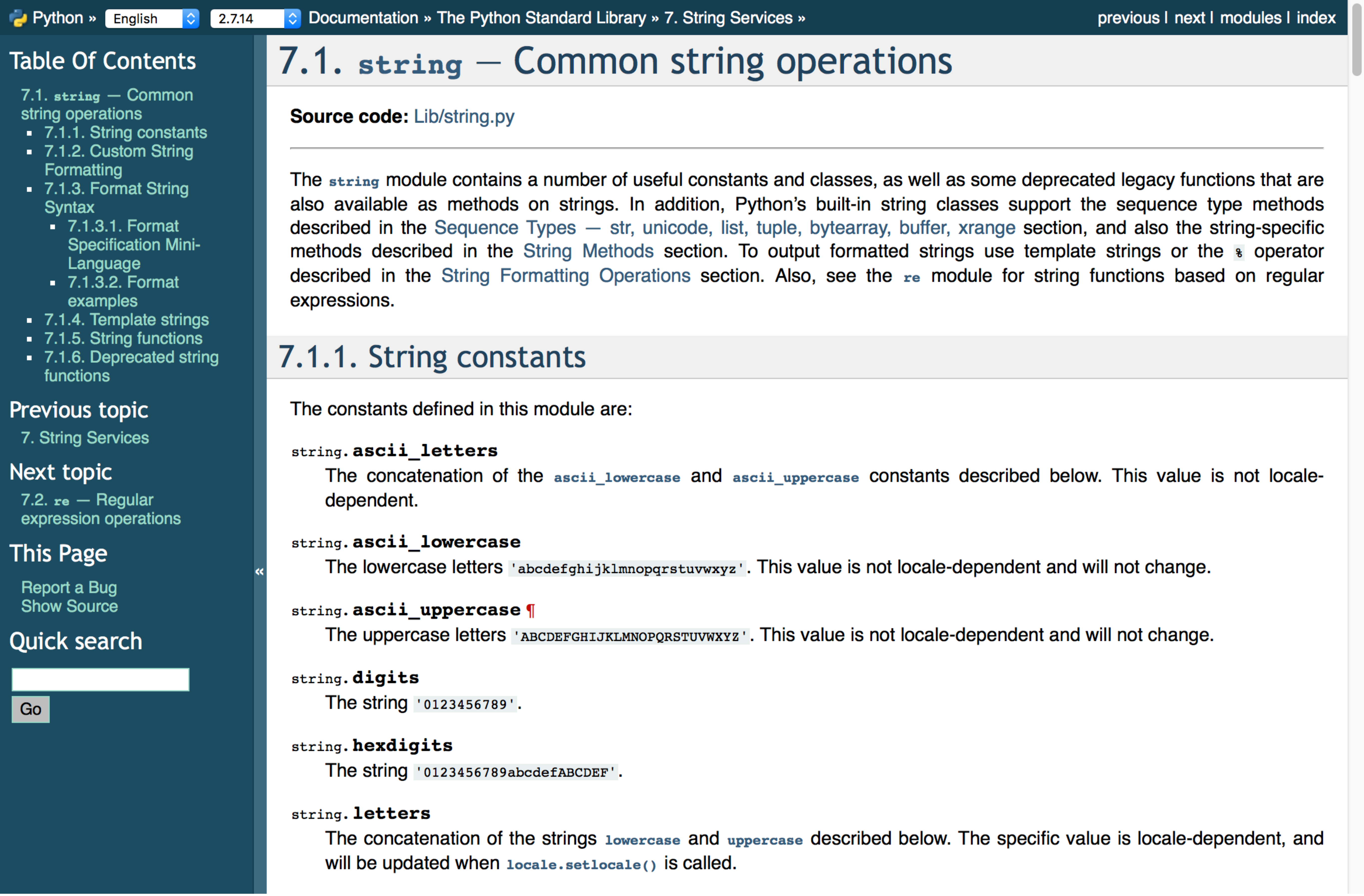
What is format in Python?

What is list comprehension in Python?

What is int () in Python?

Feedback

7.1. string — Common string operations — Python 2.7.14 documentation
https://docs.python.org/2/library/string.html ▾
A **string** containing all the characters that are considered **uppercase** letters. On most systems this is the **string** 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' . The specific value is locale-dependent, and will be updated when locale.setlocale() is called. **string**. whitespace ¶. A **string** containing all characters that are considered ...
You've visited this page 2 times. Last visit: 8/23/17



Quick search

Go

7.1. `string` — Common string operations

Source code: Lib/string.py

The `string` module contains a number of useful constants and classes, as well as some deprecated legacy functions that are also available as methods on strings. In addition, Python's built-in string classes support the sequence type methods described in the Sequence Types — str, unicode, list, tuple, bytearray, buffer, xrange section, and also the string-specific methods described in the String Methods section. To output formatted strings use template strings or the `%` operator described in the String Formatting Operations section. Also, see the `re` module for string functions based on regular expressions.

7.1.1. String constants

The constants defined in this module are:

`string.`**`ascii_letters`**

The concatenation of the `ascii_lowercase` and `ascii_uppercase` constants described below. This value is not locale-dependent.

`string.`**`ascii_lowercase`**

The lowercase letters `'abcdefghijklmnopqrstuvwxyz'`. This value is not locale-dependent and will not change.

`string.`**`ascii_uppercase`** ¶

The uppercase letters `'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`. This value is not locale-dependent and will not change.

`string.`**`digits`**

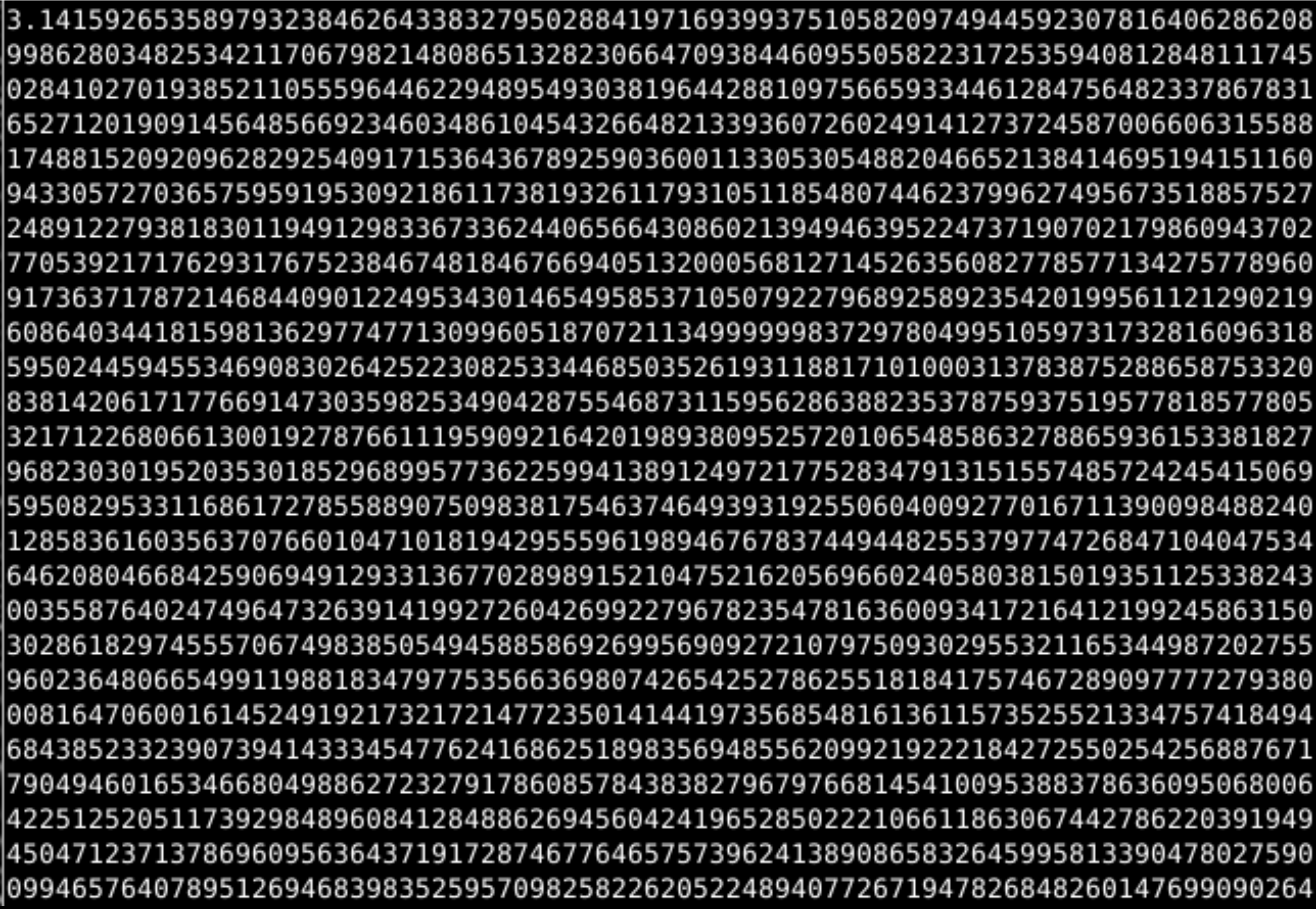
The string `'0123456789'`.

`string.`**`hexdigits`**

The string `'0123456789abcdefABCDEF'`.

`string.`**`letters`**

The concatenation of the strings `lowercase` and `uppercase` described below. The specific value is locale-dependent, and will be updated when `locale.setlocale()` is called.

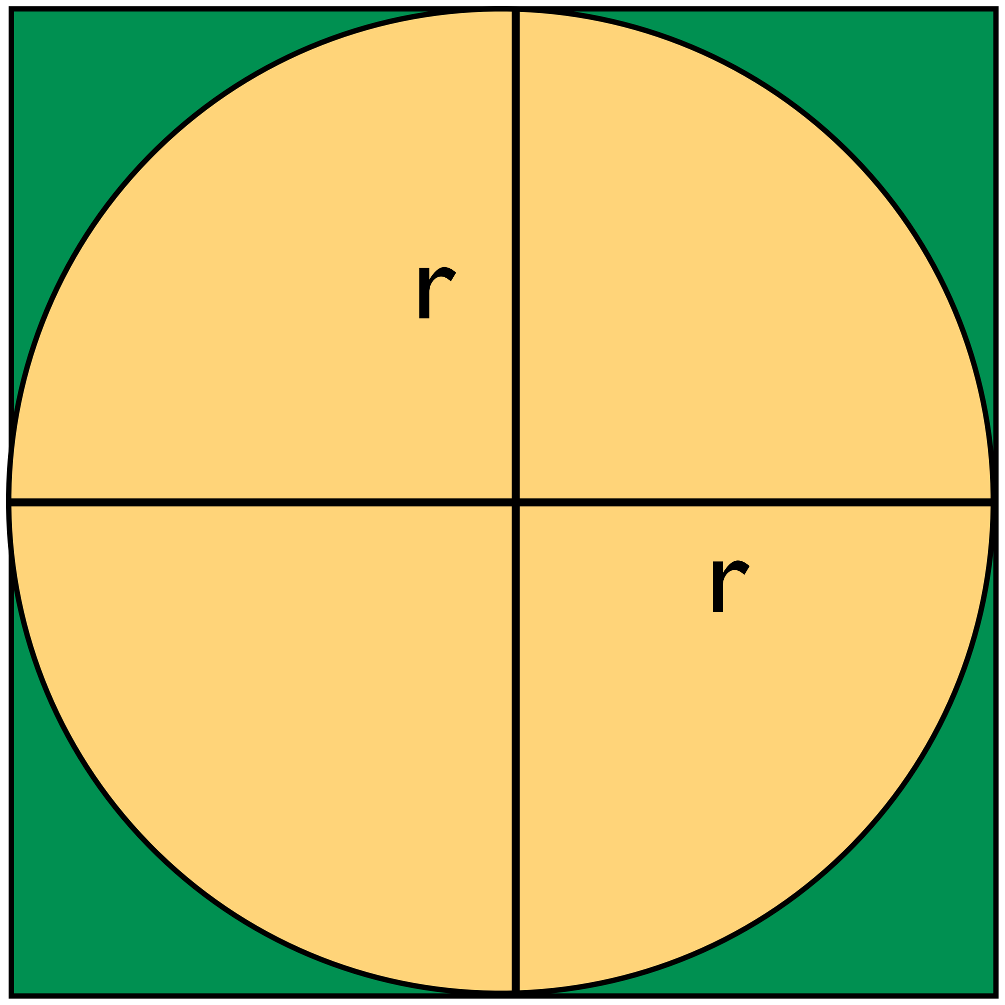


```
3.1415926535897932384626433832795028841971693993751058209749445923078164062862089
99862803482534211706798214808651328230664709384460955058223172535940812848111745
02841027019385211055596446229489549303819644288109756659334461284756482337867831
65271201909145648566923460348610454326648213393607260249141273724587006606315588
17488152092096282925409171536436789259036001133053054882046652138414695194151160
94330572703657595919530921861173819326117931051185480744623799627495673518857527
24891227938183011949129833673362440656643086021394946395224737190702179860943702
77053921717629317675238467481846766940513200056812714526356082778577134275778960
91736371787214684409012249534301465495853710507922796892589235420199561121290219
60864034418159813629774771309960518707211349999998372978049951059731732816096318
59502445945534690830264252230825334468503526193118817101000313783875288658753320
83814206171776691473035982534904287554687311595628638823537875937519577818577805
32171226806613001927876611195909216420198938095257201065485863278865936153381827
96823030195203530185296899577362259941389124972177528347913151557485724245415069
59508295331168617278558890750983817546374649393192550604009277016711390098488240
12858361603563707660104710181942955596198946767837449448255379774726847104047534
64620804668425906949129331367702898915210475216205696602405803815019351125338243
00355876402474964732639141992726042699227967823547816360093417216412199245863150
30286182974555706749838505494588586926995690927210797509302955321165344987202755
96023648066549911988183479775356636980742654252786255181841757467289097777279380
00816470600161452491921732172147723501414419735685481613611573525521334757418494
68438523323907394143334547762416862518983569485562099219222184272550254256887671
79049460165346680498862723279178608578438382796797668145410095388378636095068006
42251252051173929848960841284886269456042419652850222106611863067442786220391949
45047123713786960956364371917287467764657573962413890865832645995813390478027590
09946576407895126946839835259570982582262052248940772671947826848260147699090264
```

How to calculate π

We know that the area of a circle is

$$\pi r^2$$



Looking only at the upper right corner
we can see a green square with side r
and we can calculate the area of the square as

$$A_s = r^2$$

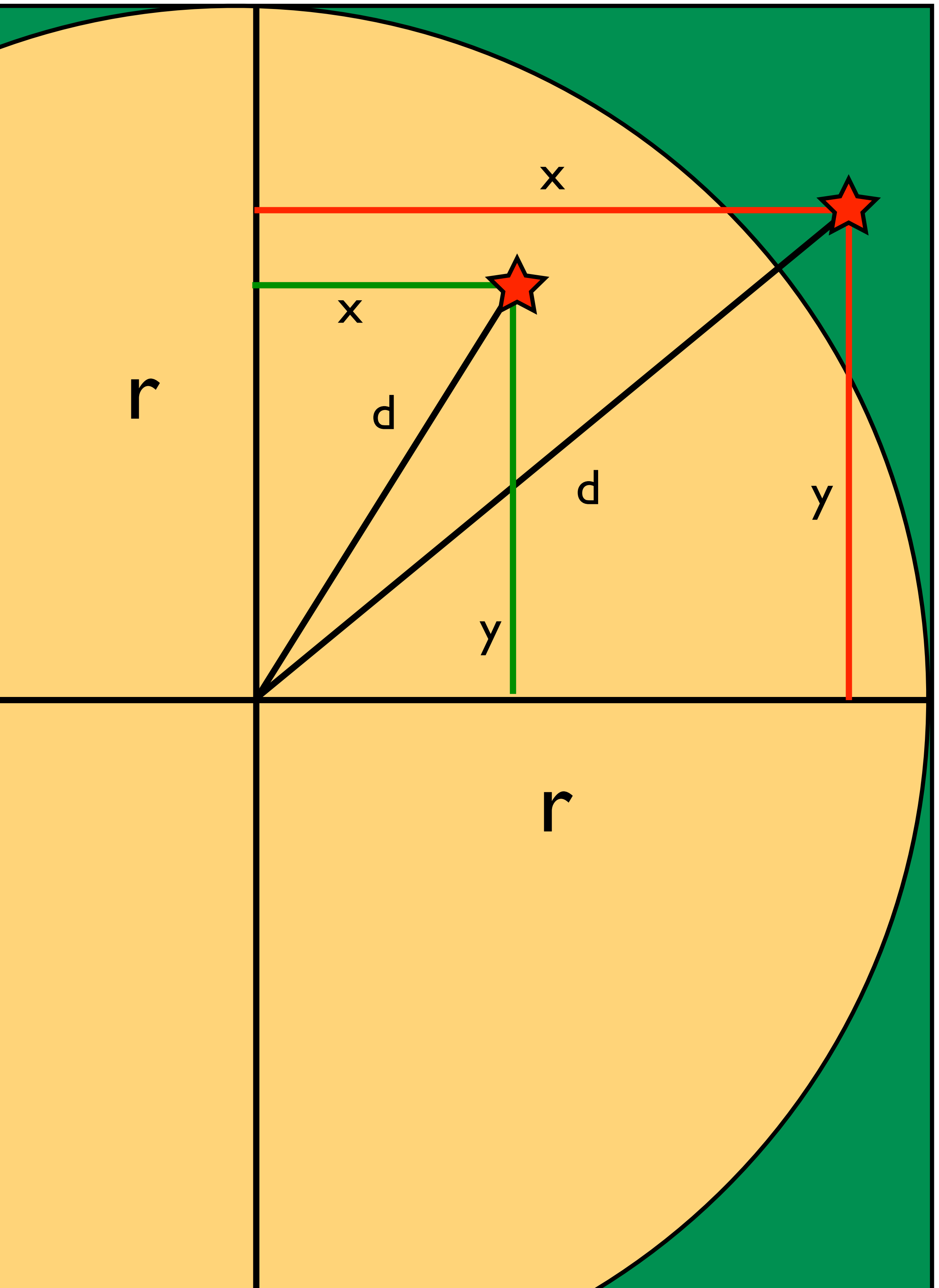
The quarter circle has the area

$$A_c = \frac{\pi}{4} r^2$$

So we can calculate the ratio of the two areas as

$$\frac{A_c}{A_s} = \frac{\frac{\pi}{4} r^2}{r^2} = \frac{\pi}{4}$$

How to calculate π



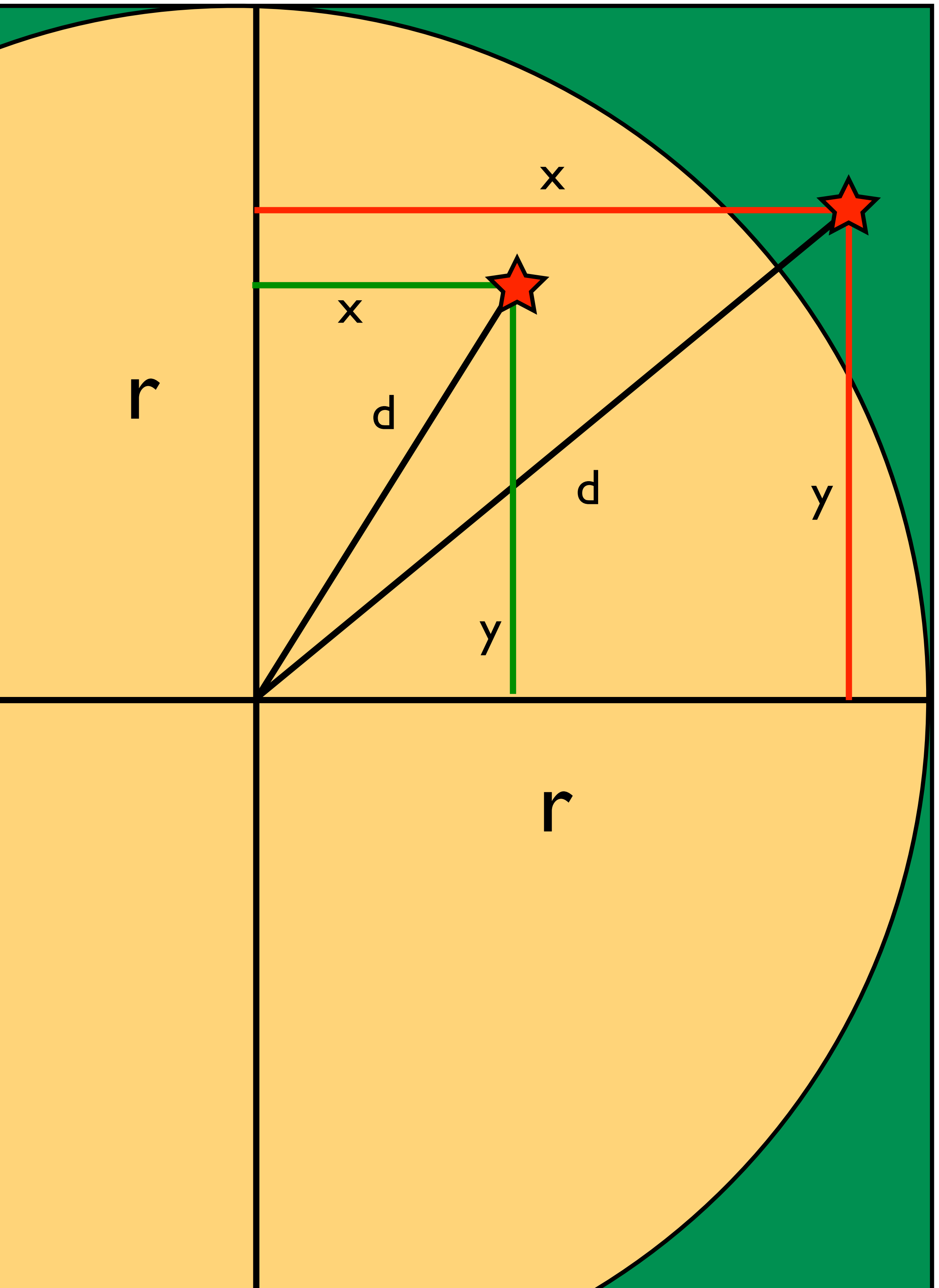
$$\frac{A_c}{A_s} = \frac{\frac{\pi}{4}r^2}{r^2} = \frac{\pi}{4}$$

The goal is now to estimate the ratio of the areas. We can devise an algorithm that draws random coordinates from the square and marks whether the coordinate fell into the circle or not. We can calculate the distance from the circle center using Pythagoras:

$$d = \sqrt{(}x^2 + y^2)$$

If d is smaller than r than we know the coordinate is in the circle otherwise only in the square. We can now create an algorithm for our program.

How to calculate π



```
# Algorithm in pseudo code
# Do many times:
#    draw x, y coordinate
#    calculate d from center
#    check whether d < r:
#       True: add 1 to circle
#       False: do nothing
#    add 1 to square
#
# print pi: ratio cicle/square * 4
```

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))
```