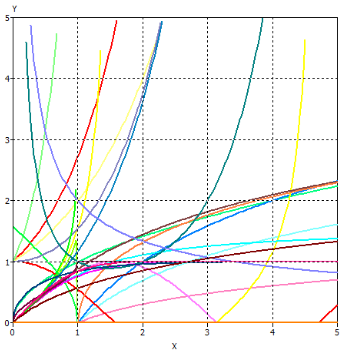
DRINK

maXbox Starter 92 – How to demystify π ?

From Shell to Hell?
HellShell or Shellboy!

This tutor explains why PI could be a normal irrational number. So DRINK stands for **D**ecimal **R**epresentation of a **I**rrational **N**umber **K**ind.
There is nothing especially or extremely interesting in the digits of a decimal representation of irrational numbers like Pi, E or the golden ratio. Or in the digits to any whole number base I think.



First we have to create a big PI (1000 digits):

```
with TBigfloat.Create1(150) do begin
  PIconst(1000)
  writeln(toString(normal)) ;
  free;
end;
```

The same we can get at:
https://www.wolframalpha.com/input/?i=pi+to+1000+digits

But how can we write a function which will return Pi (π) to a versatile given number of decimal places?

In calculus there is a thing called Taylor Series which provides an easy way to calculate many irrational values to arbitrary precision.

Pi/4 = 1 - 1/3 + 1/5 - 1/7 + ...

But this Taylor series is probably also one of the worst ways to generate PI on a computer. You have to have huge precision on your calculations and it'll take many billions of iterations to get past 3.14159.
Next try could be the use of atan, trouble is, the Taylor series for atan (PI/4 = atan(1) = 1 - 1/3 + 1/5 - 1/7 + 1/9...)converges very slowly for values of x near one, and it converges extremely slowly when x is equal to one.
So how we did it: with the AGM calculation of Pi. In 1799, Gauss was startled to discover that his **a**rithmetic-**g**eometric **m**ean connected, the half-circumference of a curve known as the lemniscate, with π, the half-circumference of a unit circle.

```
procedure TBigFloat.PiConst(const MaxSig: TMaxSig);
begin
  self.CheckPiPlaces(MaxSig);
end;

procedure TBigFloat.CheckPiPlaces(const MaxSig: TMaxSig);
var
  localPrecision: word;
begin
  localPrecision:= MaxSig+2;
  if localPrecision>zpi.sigdigits then
    calculate_PI_AGM(localPrecision+5)
  else begin
    self.Assign(zpi);
    self.RoundToPrec(MaxSig);
  end;
end;
```

Then we compare the result of the code with the wolfram alpha link above to make sure we get the right 1000 places in it:

```
   writeln('sha test1: '+sha1tohex(synsha1(tostring(normal))));
   writeln('sha test2: '+sha1tohex(synsha1(PI1000)));

   >>> sha test1: b7805c4fb1662666d7741fa8f915daacf706cd01
   >>> sha test2: b7805c4fb1662666d7741fa8f915daacf706cd01
```

Got it. So why doesn't Pi have a 0 in its first 30 digits?

3.141592653589793238462643383279502884

Let’s assume (likely but is not proven at present) that Pi is a Normal Number[1]. Amongst other things, this means that the frequency of occurrence of any digit in its decimal range is precisely 1/10. This assumption is in accord with statistical analysis of many trillions of decimal places of Pi.

We can use this to calculate the probability that there are no zeros in the first thirty digits of our **π**; In order for this to happen, zero cannot appear in the first place, a probability of 9/10 and also not in the second place, also a probability of 9/10 and so on. We get the overall probability to be:

(9/10)^30 = ~4.24% //4.23911

So it is unlikely that no zero (or another selected number) appears in the first thirty digits.
Can we say there is even less of interest in the decimal digits as a particular transcendental number, except that so many people think there is something special about it, for example you find your birth date (as 8 digits) in it.
What can we do?

1 https://peterjamesthomas.com/maths-science/the-irrational-ratio/#normal-again

We can count the frequency of each digit of Pi to assume a uniform distribution and independence and non predictability of each digit:

```
 Piconst(10000)
 writeln('zero count: '+ itoa(StrCharCount(toString(normal), '0')));
 for it:= 0 to 9 do
   println(itoa(it)+'count: '+ itoa(StrCount(toString(normal),+itoa(it)[1])));
```

```
ref: zero count: 968
0 count: 968
1 count: 1026
2 count: 1022
3 count: 976
4 count: 1012
5 count: 1047
6 count: 1023
7 count: 971
8 count: 948
9 count: 1014
```

But wait a second. Why does zero lag behind in the count. It is a perfectly non number digit? Just had to ask, its a rhetorical question. Lets check it with only 1000 digits:

```
zero count: 93
0 count: 93
1 count: 116
2 count: 103
3 count: 103
4 count: 93
5 count: 97
6 count: 94
7 count: 95
8 count: 100
9 count: 106
```

```
for it:= 0 to 9 do
    FormatF('%d count: %d',[it,StrCount(toString(normal),+itoa(it)[1])]);
```

Obviously, some digit will lag. Which digit has a reason to lag? None has a reason to lag. Zero is none. Therefore, zero lags.
-- Larry Hosken

This means as said before that each number is equally likely to be the next number so each has a 1/10 chance. Therefore, the occurrence of each digit should be equal once we reach an infinite number of decimal places.

http://www.eveandersson.com/pi/precalculated-frequencies

But Pi is obviously going to look different if we calculate it in base 8 or base 12 or any base other than 10.

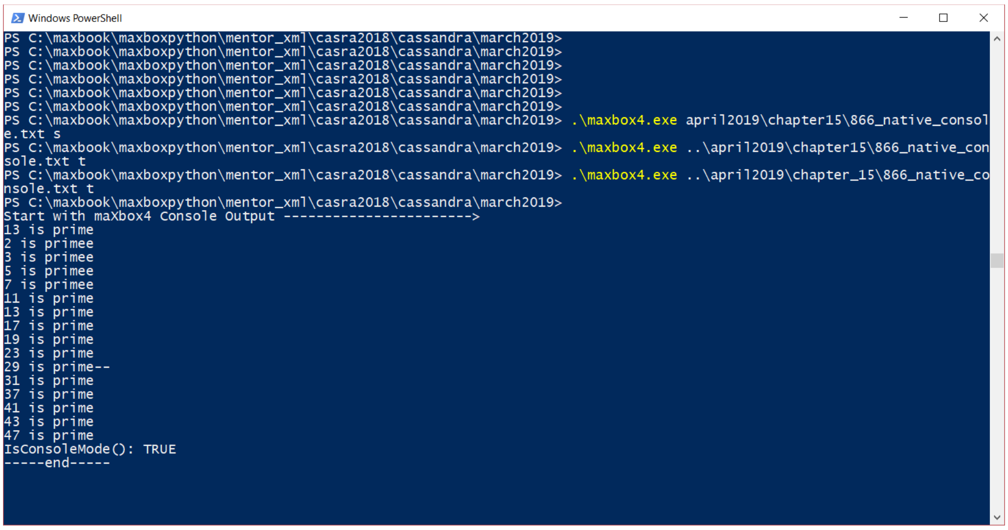
I was also reading a recent blog post by Evelyn Lamb where she mentioned in passing that 314159 is a prime number and that made me curious how many such primes there are.

https://www.johndcook.com/blog/2018/09/04/pi-primes/

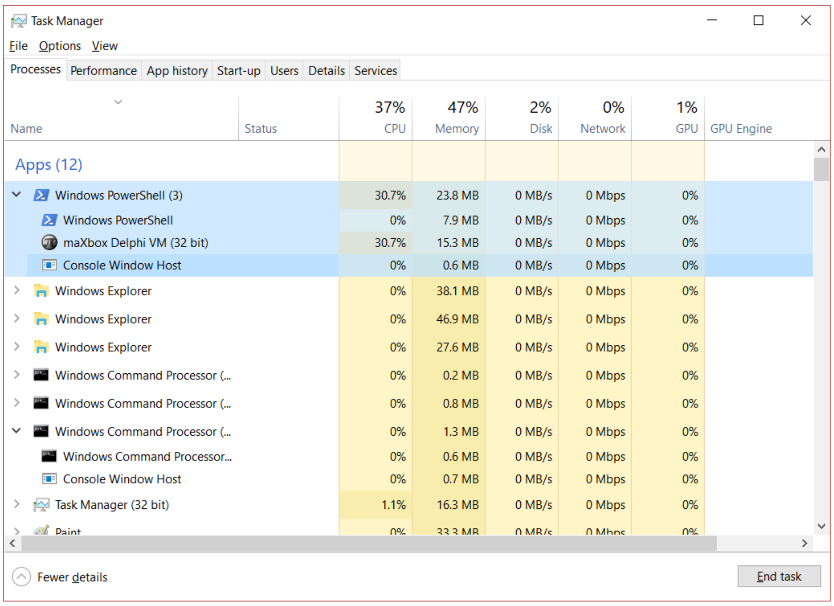
Formulas for prime-counting functions come in two kinds: arithmetic formulas and analytic formulas, see appendix prime-counting.

Prime-counting

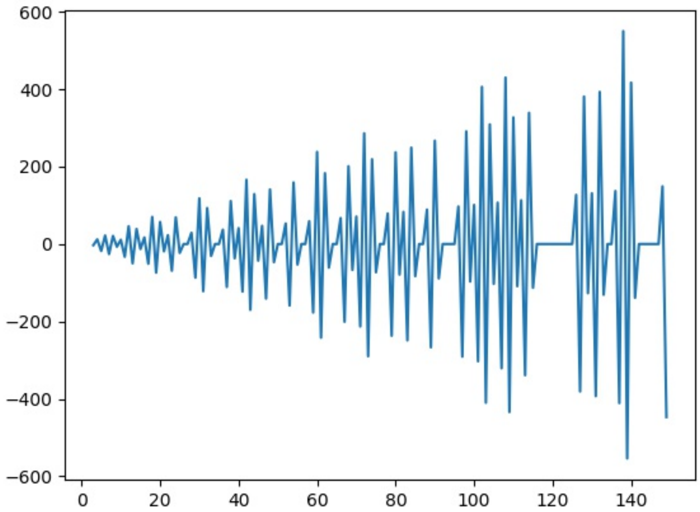
```
def is_prime(n):
  if n>1:
        divs=[k for k in range(2,n) if n%k==0]
        return len(divs)==0
  else:
        return False

def nth_prime(n):
  primes=[p for p in range(n*n+2) if is_prime(p)]
  return primes[n-1]

NativeWriteln('Start with maXbox4 Console Output--->');
   for it:= 1 to 50 do if IsPrime(it) then
                NativeWriteln(IntToStr(it)+' is prime');
NativeWriteln('-----end-----'); FreeConsole();
```



This is OK if you are just wanting to display output into the command line. But operations like redirecting output into a file for example are not working e.g.: start /wait Checker.exe > out.txt would still output into console and not into file out.txt. Different solution exists for PowerShell:



If you are lost into the source code then you could easily add parameters to your app to write output to a file instead of the console: *-o out.txt*, since it's your tool doing the writing, you can build wherever you want for example to start out of the shell and get output to the shell and in the end plot an image to another file output as a graphic like below:



Call the script from the shell with
>>> .\maxbox4.exe ..\examples\866_native_console.txt

The script can be found:

http://www.softwareschule.ch/examples/1093_XMLUtils_Tutor92tester.txt

http://www.softwareschule.ch/examples/866_native_console.txt

Author: Max Kleiner

Ref: http://www.softwareschule.ch/box.htm
https://scikit-learn.org/stable/modules/

Doc: https://maxbox4.wordpress.com

```
>>> from geopy.geocoders import Nominatim
>>> geolocator = Nominatim('maxbox-app')
>>> place,(lat,lng) = geolocator.geocode("Breitenrainplatz 2 Bern")
>>> print ("%s: %.5f, %.5f" % (place, lat, lng))
```

Release Notes maXbox 4.7.6.10 IV Jan. 2022 mX476
****
Add 25 Units + 6 Tutorials

Total of Function Calls: 35371 SHA1: of 4.7.6.10
A2B2B2D1596C6A5F3ACCED90D0C2246172A3DE2C
CRC32: 285ACBCB 31.3 MB (32,921,928 bytes)
ZIP file maxbox4.zip sha1 E267EB40AA945AD10B88EF8274C837F510DD96D4
https://www.virustotal.com/gui/file/3080a507b536ff12ec70e9a8df9eba27aca5c90709d73f42a5ad02211342bd64/details