Eine kleine Geschichte von und zu Pi

Vom Urknall zur Geburt von Pi

Die Geschichte von Pi beginnt mit einem Punkt. Ein Punkt ohne Koordinatensystem und Zeitpunkt, denn Raum und Zeit existieren noch nicht: Man kann ihn Ursprung, Null, Nil oder Singularität nennen. Physiker und Mathematiker hassen ihn, weil man ohne Raumzeit, Geometrie, Geschwindigkeit etc. kaum etwas berechnen kann, der Punkt unstetig ist und es noch nicht mal einen zweiten Punkt gibt, um wenigstens eine Gerade durch die Punkte zu ziehen.

Physiker hassen diesen Punkt vor allem, weil ohne Raum, Zeit und Bewegung ein Großteil der bekannten Naturgesetze nicht mehr funktioniert.

Aber dieser Punkt besitzt etwas: Sein “Vermögen” ist eine unvorstellbar große Menge von Energie bzw. Masse - was aber äquivalent ist, wie Einstein bewies. Außerdem wirken ausgehend von diesem Punkt zwei gegensätzliche Bestrebungen:

Massen ziehen sich durch Gravitation an und streben letztlich dazu, sich wieder in solch einem Punkt zu vereinigen, weil sie Raum und Zeit krümmen, zuvor aber erst komplexe, geordnete Strukturen, Ordnung und Information bilden - ich nenne diese Bestrebung hier Extropie weil sie der anderen Bestrebung, der Entropie entgegen wirkt.

Energie hat die Bestrebung, sich in Raum und Zeit als Strahlung auszubreiten - masselose Teilchen wie Photonen können das mit Lichtgeschwindigkeit - eine Naturkonstante unserer Welt; Massen erreichen nie Lichtgeschwindigkeit, denn sie würden dadurch unendlich schwer.

Jedenfalls siegte in unserer Welt letztlich die Entropie über die Extropie und unser jetzt “Universum” genannter Punkt breitet sich seit circa 14 Milliarden Zeit immer noch beschleunigend aus, besitzt aber lokal viele massereiche Inseln, die wieder zu schwarzen Löchern geworden sind, aber über lange Zeit wegen der Entropie immer mehr Masse verstrahlen.

Was hat das alles nun mit Pi zu tun? Pi kommt erst bei zwei Raumdimensionen ins Spiel. 2 Punkte bilden eine Gerade, 3 Punkte eventuell ein Dreieck in der Ebene und 4 Punkte einen Tetraeder im Raum.

Alle Punkte mit gleichem Abstand zu einem Ursprung bilden einen Kreis in einer Ebene oder eine Kugel im Raum - das ist die Geburt von Pi - und unser Universum. Es teilt uns heute seine explosive Geschichte durch eine weitgehend kugelförmige Hintergrundstrahlung mit einer Oberfläche von 4 * Pi * $r2$ mit.

Ein Blick in den Himmel ist ein Blick zurück bis fast zum Beginn als Punkt, denn auch mit Lichtgeschwindigkeit erreicht uns das erste Licht erst nach Milliarden Jahren.

Für unsere Zivilisation wurde Pi nicht erst mit der Erfindung des Rades, zylindrischer Behälter und allerlei Messungen und Berechnungen für Architektur, Technik und Astronomie wichtig, für die Ägypter schien der Wert 3 noch ausreichend und bis in die Neuzeit war für den Hausgebrauch der Schätzwert 3.14 genug, aber im 15ten Jahrhundert reichte das Mathematikern nicht mehr und sie fanden viele neue Rechenwege, um diese vielleicht wichtigste Strukturkonstante genauer zu bestimmen.

Der Reihe nach

So fand 1682 Gottfried Wilhelm Leibniz eine Reihenberechnung, mit der Pi sich beliebig genau berechnen lässt. Diese Leibniz Reihe

$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \ldots$.. konvergiert zu dem Wert Pi/4 - leider nur sehr langsam.
Sehr stark (exponentiell) wachsender Rechenbedarf für jede weitere Stelle macht diese Berechnung nicht sehr praktikabel.

Bereits 1655 fand John Wallis mit dem Wallisschem Produkt eine weitere Reihenberechnung, die allerdings auch nur sehr schlecht konvergiert:

$$\frac{\pi}{2} = \frac{2}{1} \times \frac{2}{3} \times \frac{4}{3} \times \frac{4}{5} \times \frac{6}{5} \times \frac{6}{7} \times \frac{8}{7} \times \frac{8}{9} \ldots.$$

Interessant bei dieser alten Rechenmethode ist, dass sie 2015 bei der Untersuchung der quantenmechanischen Analyse der Energiezustände von Wasserstoff deren Werte genau jenen der Wallis Reihe entsprachen. Ein Wasserstoff-Atom bestätigt also diese uralte Methode zur Berechnung von Pi:

https://www.scinexx.de/news/technik/klassische-pi-formel-im-wasserstoff-entdeckt/

Für bessere Rechenwege entwickeln Mathematiker bis heute immer schneller konvergierende Reihen, denn auch Computer tun sich mit Bruchrechnung und sehr großen oder kleinen Zahlen und hoher Genauigkeit schwer, wie sie bei einer Reihenberechnung auftreten. Solche großen Zahlen beanspruchen auch den begrenzten Arbeitsspeicher von Computern stark, auch weil Computerhardware für nur wenige und recht begrenzte Zahlenformate ausgelegt ist. Da ist sogar mancher Taschenrechner besser ausgestattet.

Die meisten Computer verwenden heute 64-Bit Speicherzellen und Zahlen müssen in diese hineinpassen. Einige gängige Computer-Zahlenformate zeigt folgende Tabelle:

Format	Bits	Wertebereich	Stellenzahl
Ganzzahlen vorzeichenlos	32	0 .. 4.294.967.296	9-10
Ganzzahlen mit Vorzeichen	32	-2147483648 .. 2147483647	8-9
Ganzzahlen vorzeichenlos	64	0 .. 18.446.744.073.709.551.615	19-20

Ganzzahlen mit Vorzeichen	64	-9.223.372.036.854.775.808 .. 9.223.372.036.854.775.807	18-19
Gleitkommazahl	32	+/- 3.4028235 E+/-38	6-7 😮
Gleitkommazahl	64	±4.9E-324 bis ±1.7E+308	15-17
Gleitkommazahl	128	Mantisse / Exponent unterschiedlich groß	max. 30

128 Bit - also zwei oder mehr “Speicherworte” sind heute noch selten, weil es bei den meisten Computeranwendungen eher auf Geschwindigkeit als Genauigkeit beim Rechnen ankommt.
Mathematiker und Banker bilden da eine Ausnahme.

Die geringe Stellenzahl der 32 Bit Gleitkommazahlen (float) war lange im technischen Bereich verbreitet, weil Messgeräte oft viel weniger Genauigkeit bieten, 12 Bit, also 3 ½ Stellen, sind in der Meßtechnik immer noch die Praxis.

Für die Berechnung von Pi mit vielen tausend Stellen helfen also weder Taschenrechner noch Computer-Hardware direkt weiter. Eine Lösung bieten dabei nur Ganzzahlen mit variabler Stellenzahl. Solche Zahlen kennen nicht nur Mathematik-Anwendungen wie Wolfram Mathematica, sondern vor allem Programmiersprachen, die auch bei Mathematikern und Physikern beliebt sind, wie Python, Haskell oder Lisp.

Weil heute auch die Kryptographie Zahlen mit sehr großer Stellenzahl als Schlüssel benötigt, verbreiten sich solche Zahlen als “BigInt” auch bei populären oder neuen Sprachen wie Java, C#, Dart und JavaScript zunehmend. All diese Sprachen eignen sich also, um Pi auf die Spur zu kommen und Reihen mit fast beliebiger Präzision zu berechnen.

Mathematiker bevorzugen Brüche vor Dezimalzahlen, weil sie wirklich exakte Werte darstellen können. Wenn der Divisor (Teiler) nicht eine der Basiszahlen 2 und 5 ist, kann eine Dezimalzahl wegen technisch stark begrenzter Stellenzahl bei unendlicher Ergebnis-Periodizität nicht wirklich genau sein.

Rechnet ein Computer dann gar noch binär - kennt als Basiszahl also nur die 2, liefert auch der Teiler 5 unendlich viele periodische Nachkommastellen - weshalb Bankanwendungen und die meisten Taschenrechner auch heute noch mit BCD-Dezimalzahlen rechnen, bei denen jeweils 4 Bit die Ziffern 0 bis 9 darstellen.

Zur Berechnung irrationaler Zahlen wie Wurzeln, Logarithmen, Trigonometrie und allem, wo Pi oder die Euler-Zahl eine Rolle spielen, helfen auch Brüche nicht weiter und man benötigt iterative Verfahren oder Reihen wie die Leibniz-Reihe.

Pi und das Geheimnis des Lichts

Die Verwandtschaft der Euler-Zahl e, Pi und trigonometrischen Funktionen ergibt sich auch aus der genialen “Euler Formel”:

$$e^{iy} = cos(y) + i \times sin(y)$$

Durch den Buchstaben **i** sind wir nun im Reich der komplexen bzw. imaginären Zahlen angekommen. Das i ist eigentlich ein Platzhalter der Quadratwurzel von -1. Quadriert man eine negative Zahl ist das Ergebnis immer positiv - was also ergibt $\sqrt{-1}$? Mancher Taschenrechner liefert hier das Ergebnis “keine Zahl”.

Nun, es ist recht nützlich i^2= -1 festzulegen und alle Vielfachen von i als “imaginäre” Zahlen zu definieren, wobei i als Konstante auch “imaginäre Einheit” genannt wird.

$cos(y) + i \times sin(y)$ ist also eine komplexe Zahl mit einem Real- und einem Imaginärteil.

Setzt man nun für y den Wert π ein ergibt sich:

$$cos(\pi) + i \times sin(\pi) \Rightarrow -1 + 0 = e^{i\pi} \text{ oder: } e^{i\pi} + 1 = 0$$

Diese “Eulersche Identität”genannte letzte Formel ist wirklich ein Rückgrat naturwissenschaftlich angewandter Mathematik, weil sie **e** als Basis der natürlichen Logarithmen mit Trigonometrie, komplexer Zahlentheorie und eben Pi vereinigt.

Elektrotechniker verwenden statt i den Buchstaben j für den Imaginärteil komplexer Zahlen. Wer schon mal mit Induktivitäten (Spulen) und Kapazitäten (Kondensatoren) rechnen musste, ist sicher mit komplexen Zahlen und Pi sehr vertraut.

Die gesamte Physik im Zusammenhang mit Elektromagnetismus, also elektrischen Feldern, Magnetfeldern und dem gesamten Wellenspektrum von Wechselstrom bis Gammastrahlung, basiert auf dieser Mathematik - und Pi ist in fast jeder Berechnung dabei, häufig versteckt in Form der Kreisfrequenz Omega:

$$\omega = 2\,\pi\, f \; (f \; ist \; die \; Frequenz)$$

Computer vs. unendlich große Zahlen

Doch nun zurück zur genauen Berechnung der so wichtigen Kreiszahl Pi. Abgesehen von Programmen wie Mathematica können Computer mit Bruchrechnung nicht umgehen, Gleitkommazahlen sind viel zu ungenau und Pi ist als irrationale Zahl keine Ganzzahl.

Immer mehr Programmiersprachen kennen aber einen BigInt - Zahlentyp. Für Python ist sogar das reguläre Ganzzahl-Zahlenformat nicht in der Größe begrenzt.

Nützlich zur beliebig genauen Berechnung von Pi mit BigInts sind sogenannte Spigot-Algorithmen, wie Jeremy Gibbons veröffentlichte oder Tailor Reihen, denn auch über den Arcustangens kommt man leicht an die Konstante Pi.

https://www.cs.ox.ac.uk/jeremy.gibbons/publications/spigot.pdf
https://de.wikipedia.org/wiki/Taylorreihe

Diese Algorithmen verwenden eine bei funktionalen Programmiersprachen verbreitete Generatorfunktion, die bei wiederholten Aufrufen eine noch nicht abgeschlossene Berechnung fortsetzt. Diese erkennt man an der Anweisung “yield” anstelle von “return”, um eine Funktion zu verlassen.

Das nachfolgende Python Code-Beispiel muss man nicht verstehen, der entscheidende Punkt ist der “yield” Aufruf mitten im Code statt des sonst üblichen “return” am Ende der Funktion. Mit “yield” wird die Funktion verlassen und beim nächsten Aufruf in der Folgezeile fortgesetzt. Jeder Aufruf gibt dabei die nächste Ziffer von Pi zurück. Die Generatorfunktion selbst liefert keine Primzahl, sondern einen Iterator, wie er auch für Listen zum Auslesen in einer Schleife verwendet wird.

```
def gospers_pi():
   q,r,t,n,i = 1,0,1,8,1 // Initialisierung von Termen der Reihe
   while True:
       if n == (q*(675*i-216)+125*r)//(125*t):
           yield n // wenn neue Stelle, gib sie zurück
           q,r = 10*q,10*r-10*n*t // Stellenverschiebung der Terme
       else:
           // Update der Terme für nächste Stelle
           q,r,t,i = i*(2*i-1)*q,3*(3*i+1)*(3*i+2)*((5*i-2)*q+r),\
               3*(3*i+1)*(3*i+2)*t,i+1
       n = (q*(27*i-12)+5*r) // (5*t)
```

Eine besonders einfache, aber auch schnelle Pi-Berechnung funktioniert mit Javascript und kommt ohne Generatorfunktion mit einer simplen Schleife aus. Als Javascript funktioniert sie in Web-Browsern oder einer nodejs - Befehlszeile.

Wegen der Verwendung von BigInt- Zahlen ist eine Stellen-Verschiebung (Potenzierung des Terms, der x berechnet) notwendig, um die zu berechnende Stellenzahl (BigInt Zahlen werden in Javascript durch den Suffix “n” gekennzeichnet) zu erreichen.

Die rot markierte Zahl im folgenden Code berechnet 1020 Stellen, wobei die 20 letzten allerdings wegen begrenzter Genauigkeit verworfen werden sollten, was in der letzten Zeile mit der Ausgabe geschieht.

```
let i = 1n;
let x = 3n * (10n ** 1020n);
let pi = x;
while (x > 0) {
       x = x * i / ((i + 1n) * 4n);
```

```
        pi += x / (i + 2n);
        i += 2n;
}
console.log(pi / (10n ** 20n));
```

Eine Pi berechnende Pi Pico - Maschine zum Selber Bauen

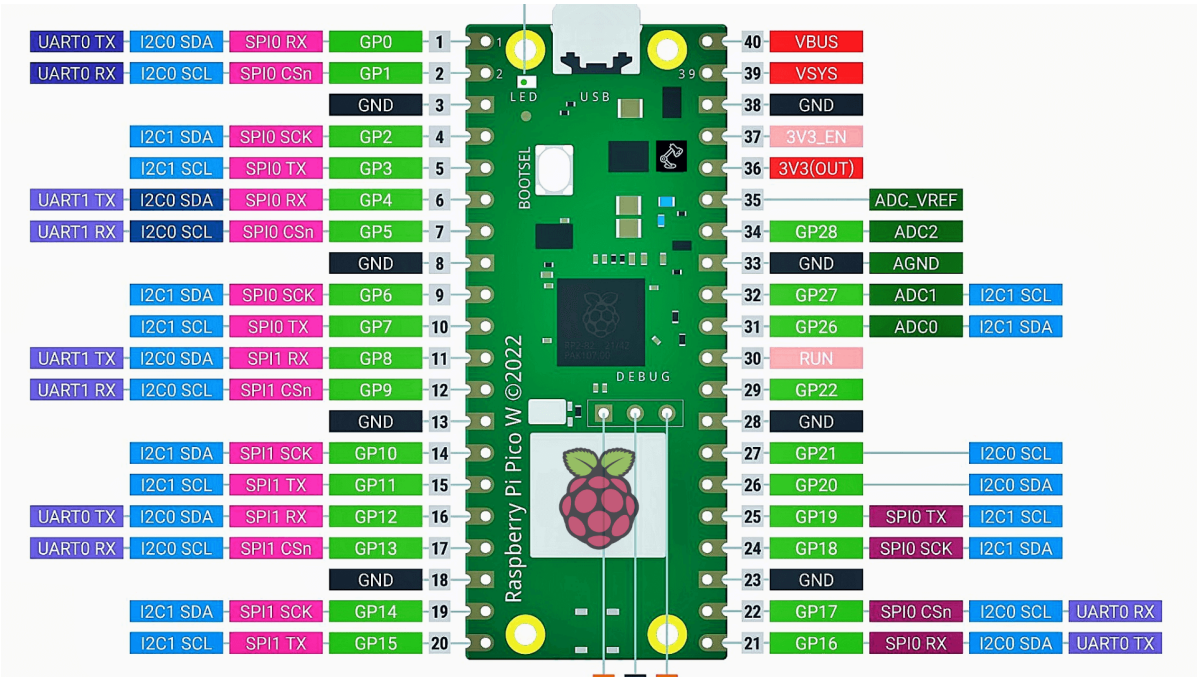
Es gibt wohl kaum einen Bereich technischen Fortschritts, der sich schneller entwickelt hat als die Mikroelektronik. Aktuelle Smartphones leisten mehr als die Supercomputer der 1980er. Konkret hängt bereits ein altes iPhone4 locker einen Cray2 Supercomputer von 1985 ab.

Auf die Mikroprozessoren der 1980er folgten die Micro-Controller und SoCs (System on a Chip) heute. SoCs enthalten nicht nur das Prozessor genannte Rechen- und Steuerwerk eines Computers, sondern auch gleich Arbeitsspeicher, etwas Massenspeicher und vielfältige Peripherie, wie Ein/Ausgabe und Netzwerkverbindungen via WLAN oder Bluetooth. Der Konkurrenzdruck im Smartphone- und Gadget-Bereich macht das bei immer weiter fallenden Stückpreisen möglich.

Hier beschreibe ich den Aufbau eines mit interaktiv nutzbaren Python-Interpreter ausgestatteten “Homecomputers”, der über ein LED-Display eine Laufschrift ausgeben kann (wie Stellen von Pi).
Das Projekt ist ein einfacher Einstieg in die Welt der Mikrocontroller, denn es werden lediglich 2 Bausteine, ein Experimentierboard (Breadboard) und etwas Schaltdraht für Verbindungen benötigt. Ein Computer mit USB-Anschluss muss für die Programmierung des Controllers aber vorhanden sein.

Der Raspberry Pi Pico Microcontroller

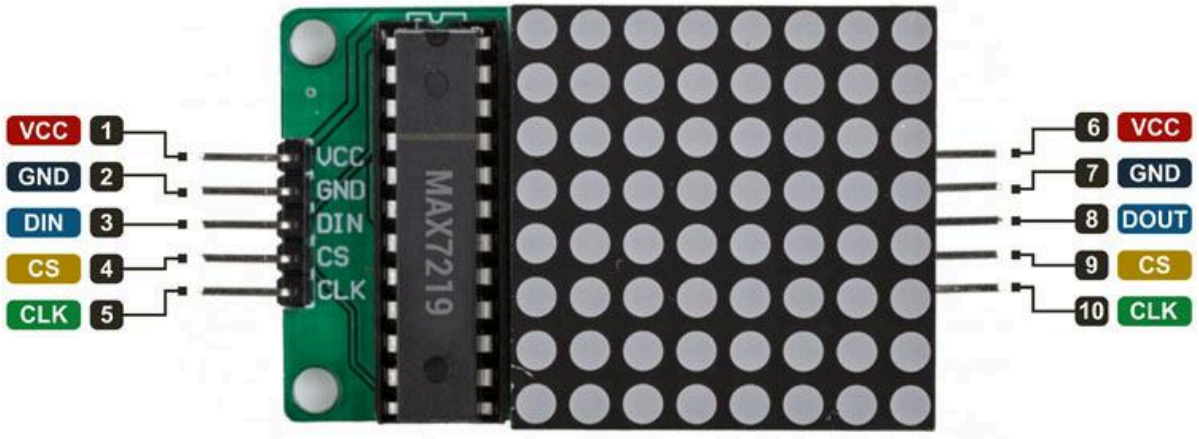
https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html



Der Pico ist der kleinste Computer der vor allem an Heimanwender gerichtete Raspberry Pi Serie. Er basiert auf einem RP2040 SoC, der für Breadboard-freundlichen Aufbau auf einer Platine mit 40 Anschlüssen gelötet ist und sonst nur wenige Komponenten - etwa zur Spannungsversorgung - besitzt. Das Anschlussbild oben soll nur zeigen, dass die meisten Anschlüsse bis zu 4 Bedeutungen haben, die durch Programmierung festgelegt werden (GP steht für “general purpose”), also universeller Verwendungszweck. Zwei mit 133 MHz getaktete ARM Cortex M0+ Rechenkerne, 268 KB Hauptspeicher und 2 MB Massenspeicher erinnern zwar eher an PCs der späten 80er, aber übertreffen immer noch damals noch verbreitete Home Computer. Das reicht aber, um einen aktuellen Python 3 Interpreter interaktiv verwenden zu können, der viel mehr kann als jeder BASIC Home Computer damals. Vor allem kann man sehr viel mehr damit steuern. Außerdem kostet dieses Kit aktuell weniger als 5€.

Stromversorgung, Dialog und Programmierung erfolgen über den USB-Anschluss.

Max 7219 LED Matrix Display Controller
https://www.analog.com/media/en/technical-documentation/data-sheets/max7219-max7221.pdf



Zwar kann man einzelne LEDs auch direkt an die GP-Ausgänge des Pi Pico anschließen, aber viele davon brauchen auch viel Strom und erfordern zusätzlich noch viele Widerstände und Kabel. Dafür gibt es mit dem MAX7219 einen speziellen Controller der bis zu 64 LEDs multiplexen und mit Strom versorgen kann, sodass nur 5 Leitungen mit dem Pico verbunden werden müssen:

Anschluss	**Bedeutung**
VCC	5 Volt Stromversorgung
GND	Masse (Null-Leiter)
Data In	Datenleitung
CS	Auswahl eines Max7219
CLK	Taktleitung

Das Beste ist aber, dass diese Module kaskadiert werden können und im Handel auch fertige Streifen mit 4 oder 8 dieser Module zu bekommen sind - um etwa Laufschrift-Anzeigen zu realisieren. Dazu werden die CS Leitungen der Module miteinander verkettet (daisy-chain), auch mehrere 4er Displays können so noch verbunden werden, wenn größere Displays benötigt werden. Ein weiterer Vorteil dieser Module ist, dass jede LED auch einzeln adressierbar ist und man so auch eigene Zeichen und Zeichensätze bauen kann.
Vierer-Module mit 4*64 LEDs kosten meist unter 10€.

Verdrahtung der Schaltung

Da letztlich nur 5 Leitungen zwischen dem ersten MAX7219 Modul und dem Pi Pico angeschlossen werden müssen, ist ein “Aufbau” dieses Computers denkbar einfach:

Raspberry Pi Pico	**erster MAX7219**	**Kommentar**
Pin 4 SPIO SCK	CLK	Taktausgang Pi Pico
Pin 5 SPIO TX	Data In	Datenausgang Pi Pico
Pin7 SPIO CS	CS	Select Ausgang Pi Pico
Pin 38	GND	Null Leiter
Pin 40 VBUS	VCC	5V von Pi Pico USB Eingang

Python Quellcode für die Laufschrift-Ausgabe der berechneten Stellen von Pi

Das gesamte Projekt kann von meinem Github - Account geladen werden:
https://github.com/ambotaku/get_pi.git

Um dies Dokument nicht unnötig aufzublähen, veröffentliche ich hier nur das Hauptprogramm, welches die eingangs schon beschriebene Python Generatorfunktion gospers_pi.py nutzt, um Pi-Ziffern als LED Laufschrift anzuzeigen:

```
import max7219                          # display controller
library
from machine import Pin, ADC, SPI    # pico hardware interfaces
from time import sleep
from gospers_pi import gospers_pi    # Pi generator function

def get_pi():
   pi = gospers_pi()    # get generator instance

   # setup SPI interface pins
   spi = SPI(0, baudrate=10000000, polarity=1, phase=0,
sck=Pin(2),\ mosi=Pin(3))

   ss = Pin(5, Pin.OUT)     #optional ticker speed
potentiometer out
   adc0 = ADC(0)            #potentiometer analog input

   #initialize 4 chained max7219 modules
```

```
display = max7219.Matrix8x8(spi, ss, 4)
display.brightness(1)    # display brightness (1-15)
display.fill(0)          # clear display
display.show()           # draw display

tx = ' '*4               # setup digits to display

while True:
    value = adc0.read_u16() # read speed potentiometer

    display.fill(0)          # clear display
    tx=tx[1:]                # remove first digit
    tx += str(next(pi))      # add next digit as last
    display.text(tx,0,0)     # store all digits
    display.show()           # update display
    sleep(.1 + value/65535) # pause scrolling
```