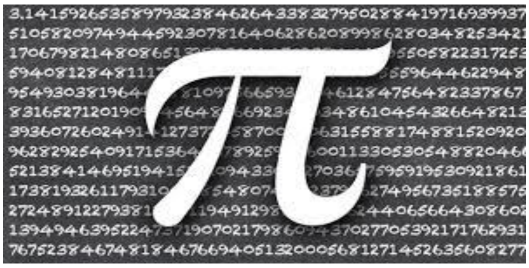
PIE_ROM Manual
HP-41 Module



Introduction and Credits.

This HP-41 module provides a short collection of functions and routines dedicated to the two most-famous irrational numbers in math: number pi and number e. With just a 10-digit mantissa capability the HP41 platform surely isn't the natural choice for ground-breaking, never-before covered methods and approaches to the calculation of these numbers – remember: our trusty Coconut "believes that π is a rational number equal to 104348/33215). Nevertheless, there's still room for interesting exercises and ingenious approaches to work-around such platform limitations.

Several MCODE functions and short FOCAL routines are provided mainly as programming exercises; that is application examples using general techniques like Continued Fractions or making use of other fields like integration, random numbers and nested radicals – always applied to the pi/e subject.

In the "-Pi DIGITS" section the module includes all relevant programs on this subject known to the author published in different magazines, books, and forums – in what should be a comprehensive archive of available material on this topic. In particular the MCODE function **MDOP** written by Peter Platzer, is a remarkable implementation even if it requires Q-RAM to hold the results, so dust off your HEPAX RAM for the task.

In terms of the sources used, the usual suspects are to blame: PPC Journals (see Ron Knapp's classic programs), application books and user forums. *Very special thanks to Valentín Albillo* for his seminal and always original contributions along the years, a real powerhouse on this and many other math subjects. Many thanks to Gerson W. Barbosa, Jean-Marc Baillard, Thomas Klemm, Benoit Maag and everybody contributing to the MoHP forum on this subject. As a wise man once said, "*if something works as expected it's their credit, if it doesn't it's my fault*".

Dependencies.

Lastly, note that some programs use functions from the SandMath – which in turn needs the Library#4 as well. This dependency is more than justified to enable the venerable 41 platform to use RCL math functions (for direct compatibility with HP-42 code); and to apply off-the-beaten-path approaches using hyperbolic functions, CROOT solver, AGM and FLOOR, as well as to benefit from the remarkable Continued Fractions MCODE implementation written by Greg McClure, also available in that module.

General references:

https://en.wikipedia.org/wiki/Approximations_of_%CF%80#Gregory%E2%80%93Leibniz_series
https://mathworld.wolfram.com/PiApproximations.html

Without further ado, here is a list of the functions in the Main FAT table.

XROM#	Function	Description	Author
09.00	**-PI/E ROM**	Section Header	*n/a*
09.01	**“Σ3PI**	Madhava Alternating Series	*Thomas Klemm*
09.02	**“GBPI**	Gerson’s Pi formula	*Barbosa-Martin*
09.03	**E2PI**	From e to π	*Á. Martin*
09.04	**LIUHUI**	Liu Hui’s Pi formula	*Á. Martin*
09.05	**“LNPI**	Ramanujan Ln-based π formula	*Á. Martin*
09.06	**“MCE**	Monte-Carlo method for e	*Albillo-Martin*
09.07	**“MCPI**	Monte-Carlo method for π	*Albillo-Martin*
09.08	**“MYPI**	10-digit π using an AGM closed-form	*Á. Martin*
09.09	**“PICUBE**	π from cubic equation root	*Albillo-Martin*
09.10	**PI2E**	From π to e	*Á. Martin*
09.11	**“PIFL**	π using a FLOOR loop	*Valentín Albillo*
09.12	**PISIN**	π using a SIN loop	*Á. Martin*
09.13	**PPIE**	Valentín’s Product formula w/ correction	*Á. Martin*
09.14	**RAMA10**	Ramanujan formula (10-digit accuracy)	*Á. Martin*
09.15	**“SBPI**	Salamin-Brent Algorithm – based on AGM	*Á. Martin*
09.16	**“VAPI**	π using a corrected Leibnitz series	*Valentín Albillo*
09.17	**VIETA**	Viete’s formula	*Á. Martin*
09.18	**WALLIS**	Wallis formula (n in X)	*Á. Martin*
09.19	**“WPI**	Wallis formula – V2	*JM Baillard*
09.20	**“WPIH**	Wallis formula w/ hyperbolics	*Werner*
09.21	**“WWPI**	Wallis-Wasicki Formula	*Gerson W. Barbosa*
09.22	**-PIE DIGITS**	*Section header*	*n/a*
09.23	**EB**	Erdós-Borwein constant	*Á. Martin*
09.24	**IROUND**	Integer Round	*Á. Martin*
09.25	**MDOP _ _ _”_**	Many Digits of π – Spigot algorithm	*Peter Platzer*
09.26	**“PI1K**	π to 1,000 digits	*Ron Knapp*
09.27	**"E2900**	E to 2,900 digits	*Ron Knapp*
09.28	**“R**	Result output	*Ron Knapp*
09.29	**“PIDIG**	π up to 1,590 digits	*Jean-Marc Baillard*
09.30	**“EZHAL**	E to 1,143 digits	*Eckard Gehrke*
09.31	**“PIZHAL**	π to 800 digits – Machin’s method	*Eckard Gehrke*
09.32	**“OUT**	Output results	*Eckard Gehrke*
09.33	**“Σ2PI**	π digits	*Benoit Maag*
09.34	**-CONT FRAC**	*Section header*	*n/a*
09.35	**“CFE**	Continued Fractions for e	*Martin-McClure*
09.36	**“*E**	Auxiliary for “CFE	*Á. Martin*
09.37	**“CFPI**	Continued Fractions for π	*Martin-McClure*
09.38	**“P0**	Auxiliary for “CFPI	*Á. Martin*
09.39	**“CFP1**	Continued Fractions for π – version 1	*Martin-McClure*
09.40	**“P1**	Auxiliary for “CFP1	*Á. Martin*
09.41	**“CWPI**	Wallis-adjusted CF for π	*Martin-McClure*
09.42	**“*WP**	Auxiliary for “CWPI	*Á. Martin*
09.43	**“PITG**	π by simple integration	*Á. Martin*
09.44	**“*I**	Integrand function	*Á. Martin*

Pi Approximations

The module includes a few short functions based on well-known pi approximations. There are literally hundreds of them (see for instance *Pi Approximations -- from Wolfram MathWorld*) but I've chosen those meaningful to the HP-41 platform in terms of decimal digits and somewhat the available function set and CPU speed.

Function	Description	Input	Output
LIUHUI	Liu Hui's formula	none	3,141590529
RAMA10	Ramanujan formula (10-digit)	none	3,141592654
E2PI	From e to π	none	3,141592653
PI2E	From π to e	none	2,718281828
PISIN	SIN-based iterations	none	3.141592654
VIETA	Viete's formula	none	3,141592654
"PICUBE	π as root of cubic equation	none	3.141592654
"PIFL	FLOOR-based iterations	n in X	Function of n
"PITG	INTEG-based calculation	FIX-9	3.141592654
"Σ3PI	Madhava Series	none	3.141592654
" GBPI	Gerson's formula (e-based)	none	3.141592654

They're described below.

- **RAMA10** uses one of the many Ramanujan's approximations of pi, correct to 10 decimal digits. It requires no input. The result is placed in X and the stack is lifted (unless CPU F11 is clear)

$$\pi \approx \approx \frac{355}{113}\left(1 - \frac{0.0003}{3533}\right)$$

XEQ "RAMA10" => 3,141592654 (in FIX 9)

- **LIUHUI** uses Liu Hui's formula to calculate an approximation of pi, correct to 5 decimal digits. It requires no input. The result is placed in X, the stack is lifted (unless CPU F11 is clear)

$$\pi \approx 768\sqrt{2-\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+1}}}}}}}}}$$
$$\approx 3.141590463236763.$$

XEQ " LIUHUI" => 3,141590529 (in FIX 9)

- **VIETA** uses Viete's formula for the calculation, a more accurate one in that is returns a correct value to the 11th. decimal digit (although this is not taken advantage of on the HP-41 or course).

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{2}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \cdots$$

The FOCAL program listed below would be equivalent to the MCODE implementations of VIETA and LIUHUI. No data registers are used but ALPHA registers M,N are needed. Refer to the appendix section of the manual for the details on the MCODE implementation.

1	LBL "VIETA"		1	LBL "LIUHUI"	
2	*E*		2	*8*	*# ot iters*
3	STO M	*initial term*	3	ENTER^	
4	STO N	*initial result*	4	*E*	*initial value*
5	LBL 00		5	LBL 00	
6	RCL M		6	*2*	
7	0	*loop result*	7	+	
8	LBL 01		8	SQRT	
9	*2*		9	DSE Y	
10	+	*add to previous*	10	GTO 00	
11	SQRT	*square root*	11	CHS	final term
12	DSE Y	*repeat loop term*	12	*2*	is negative
13	GTO 01	*until all done*	13	+	
14	2	*divide by 2*	14	SQRT	
15	/		15	768	
16	RCL N	*partial product*	16	*	
17	*	*updated*	17	END	
18	FS? 10				
19	VIEW X	*show if F10 set*			
20	X<>N				
21	RCL N				
22	-	delta			
23	X=0?	*delta=zero?*			
24	GTO 02	*yes, exit*			
25	ISG M	*do next term*			
26	NOP				
27	GTO 00				
28	LBL 02				
29	RCL N				
30	1/X				
31	ST+ X				
32	CLD				
33	END				

The next three functions are taken from one of Valentín Albillo's famous challenges (see: "*HP Challenge VA511 - 2020-03-14 - SRC 006 Pi Day 2020 Special*"),

The Function **PISIN** uses a SIN-based iterative method to estimate p. The method is a very simple one, and it's also highly efficient: starting with the value 3, only three iterations already achieve a 10-digit accuracy in the result.

See on the right the short & sweet user code routine (who said FOCAL wasn't efficient?), which is equivalent to the MCODE code implemented in the module, also shown below for your reference.

1	LBL "PISIN"	
2	RAD	
3	3	*initial value*
4	ENTER^	*also # of iters*
5	LBL 00	
6	SIN	
7	LASTX	
8	+	*x+sin(x)*
9	DSE Y	
10	GTO 00	*do next*
11	RTN	done.

XEQ "PISIN" -> 3.141592654

Header	A80B	*08E*	*"N"*	
Header	A80C	*009*	*"I"*	***SIN-based iterations***
Header	A80D	*013*	*"S"*	
Header	A80E	*009*	*"I"*	
Header	A80F	*010*	*"P"*	*Martin-Albillo*
PISIN	**A810**	**18C**	**?FSET 11**	
	A811	*3B5*	*?C XQ*	*Stack lift*
	A812	*051*	*->14ED*	*[R_SUB]*
	A813	2A0	SETDEC	
	A814	04E	C=0 ALL	
	A815	35C	PT= 12	***C=3***
	A816	0D0	LD@PT- 3	
	A817	0E8	WRIT 3(X)	*initial value*
	A818	0E0	SLCT Q	
	A819	01C	PT= 3	*will loop three times*
LOOP3	A81A	0A0	SLCT P	
	A81B	070	N=C ALL	*required by [TRG100]*
	A81C	*3C4*	*ST=0*	*skips [TRGSET]*
	A81D	*048*	*SETF 4*	***result in RAD***
	A81E	*229*	*?NC XQ*	*it uses [SCR] as well*
	A81F	*048*	*->128A*	*[SIN1]*
	A820	**11E**	**A=C MS**	*bug or what?*
	A821	0F8	READ 3(X)	
	A822	*025*	*?NC XQ*	
	A823	*060*	*->1809*	*[AD1_10]*
	A824	0E8	WRIT 3(X)	
	A825	0E0	SLCT Q	
	A826	3D4	PT=PT-1	*next iteration*
	A827	394	?PT= 0	*all done?*
	A828	*393*	*JNC -14d*	*[LOOP3]*
	A829	*3C1*	*?NC GO*	*Normal Function Return*
	A82A	*002*	*->00F0*	*[NFRPU]*

- The routine **PIFL** is based on a FLOOR algorithm. Although it shares with the previous one to be short in code length, its efficiency is drastically worse: it takes quite a large number of iterations to achieve a decent accuracy, as the table below shows. For obvious reasons a TURBO-50 CL or better yet, V41 in turbo mode are recommended.

# of terms	Result
10	3.030303030
100	3.092145949
1,000	3.139027529
10,000	3.141331981
100,000	3.141528892

1	LBL "PIFL"	
2	STO 00	*n*
3	E	
4	-	*n-1*
5	2 E-3	*2.003*
6	+	*(n+1).003*
7	RCL 00	*n*
8	LBL 01	
9	RCL Y	*k,003*
10	INT	*k*
11	CHS	*-k*
12	/	*-n/k*
13	LASTX	*-k*
14	X<>Y	
15	FLOOR	*floor(-n/k)*
16	*	*-k*floor(-n/k)*
17	DSE Y	*n=n-1*
18	*GTO 01*	
19	1/X	*1/result*
20	RCL 00	*n*
21	X^2	*n^2*
22	*	*n^2 / Result*
23	END	*done.*

- On the other hand, **PICUBE** uses a "tuned" cubic equation as the basis for the calculation. It is quite fast as no iterations are needed and because it uses the SandMath's **CROOT** (in MCODE) to obtain the real root of the equation.

Let xo be the real root of:

$$x^3 - 6x^2 + 4x - 2 = 0$$

then:

$$\pi = 24. \, \mathrm{Ln}(x0) / \mathrm{sqrt}(163)$$

XEQ "PICUBE" => 3.141592654

1	LBL "PICUBE"	
2	*E*	*enter the three*
3	ENTER^	*coefficientes*
4	*-6*	
5	ENTER^	
6	*4*	
7	ENTER^	
8	*-2*	
9	CROOT	
10	RDN	*discard the twol*
11	RDN	*non-real roots*
12	LN	
13	*24*	*do the math*
14	*	*to end.*
15	*163*	
16	SQRT	
17	/	
18	END	

Pi using Madhava Alternating Series - $\boxed{\Sigma 3PI}$

See *https://www.hpmuseum.org/forum/thread-18129.html*

The series expression is as follows:

$$\frac{\pi}{6} = \frac{1}{\sqrt{3}}\left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots\right)$$

An interesting expression by itself that proves to be elusive in its implementation due to its alternating character – one of the known weak points of this computing platform.

Fortunately, Thomas Klemm provided a capable HP-42 version that has been added to the ROM. I've pre-set the number of terms to 43, as per his findings in the thread referenced above.

```
01 ▸LBL "Σ3PI"
02  43
03 ▸LBL 00
04  1/X
05  LASTX
06  X<> ST Z
07  3
08  ÷
09  -
10  X<>Y
11  2
12  -
13  X>0?
14  GTO 00
15  R↓
16  END
```

XEQ "Σ3PI" => 3.141592653

Another Ramanujan formula to end this section:

$$\frac{\ln\left\{[2 \times 5! + (8-1)!]^{\sqrt{9}} + 4! + (3!)!\right\}}{\sqrt{67}}$$

A undeniably beautiful approximation of pi, easily programmed as follows:

```
01  LBL "LNPI"
02  7
03  FACT
04  5
05  FACT
06  ST+ X
07  +
08  9
09  SQRT
10  Y^X
11  4
12  FACT
13  +
14  3
15  FACT
16  FACT
17  +
18  LN
19  67
20  SQRT
21  /
22  END
```

Merry-go-Round: From pi to e and back again.

The pair of functions below make use of the expressions linking e and pi to obtain one when the other is known – albeit in a not-so-trivial way; which BTW would be the Euler "identity" (to loosely use the term) relating pi, e, and i in the famous equation "e^(iπ)-1=0"

isolating π -> π = Ln(-1) / i, and isolating e -> e = (-1)^(1/iπ) ;

which on the 41Z is a trivial, easy as a pie, two mini-programs (5- and 7-steps respectively):

{ LBL "ZPIE", -1, ZREAL^, ZLN, Z/I, ZAVIEW, END }
{ LBL "ZEPI", -1, ZREAL^, PI, ZIMAG^, ZINV, W^Z, ZAVIEW, END }

XEQ "ZPIE" => 3,141592654+J0
XEQ "ZEPI" => 2,718281828+J0

But we're digressing, let's bring the conversation back to the PIE_ROM, shall we?

<u>From pi to e:</u>

Simply making use of the series definition of the exponential function, calculated for x= π:

$$\exp x := \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

; thus:

π = Ln (1+ π^2 /2 + π^3 /6 + π^4 /24 + π^5 / 120 + . . .)

Which converges moderately fast, so with about 22 terms we reach the 10-digit accuracy sought for.

Using **PI2E** does not require any input, and as expected will place the result in X after lifting the stack:

PI2E => 2,718281828

<u>Conversely, from e to pi:</u>

Here we're using the formula below:

$$\pi = 4(\arctan \mathrm{e} - \arctan \frac{\mathrm{e}-1}{\mathrm{e}+1})$$

Using **E2PI** does not require any input, and as expected will place the result in X after lifting the stack:

E2PI => 3,141592653

A FOCAL program listing equivalent to the MCODE functions included in the module is given next –.

1	LBL "E->PI"	23	LBL "PI->E"
2	LBL A	24	LBL B
3	RAD	25	E
4	E	26	ENTER^
5	E^X	27	LBL 00
6	ENTER^	28	PI
7	ATAN	29	RCL Z
8	X<>Y	30	Y^X
9	ENTER^	31	LASTX
10	ENTER^	32	FACT
11	E	33	/
12	-	34	RND
13	X<>Y	35	X=0?
14	E	36	GTO 02
15	+	37	+
16	/	38	ISG Y
17	ATAN	39	NOP
18	-	40	GTO 00
19	4	41	LBL 02
20	*	42	X<>Y
21	RTN	43	PI
22	GTO A	44	1/X
		45	Y^X
		46	RTN
		47	GTO B
		48	END

Gerson Barbosa has contributed another way to calculate π from e, using his own formula shown below, that has been programmed in the straightforward **GBPI** routine as follows:

```
01  LBL "GBPI"
02  E
03  E^X
04  -12
05  Y^X
06  5.6789
07  +
08  12
09  1/X
10  Y^X
11  E
12  E^X
13  *
14  END
```

$$e \times \sqrt[12]{e^{-3\times4} + 5.67890}$$

XEQ "GBPI" => 3.141592654

Not sure where this formula came from but sure enough it does the job with flying colors, thanks Gerson!

Pi in the Sky – The flying squad.

And completing this section we have yet another very recent, Valentín's 2022 Pi Day contribution – *https://www.hpmuseum.org/forum/thread-18110.html*

In it Valentín introduces an original expression also linking the values of pi and e, and furthermore, he provides up to four correction factors to improve on the results from the product formula, stating that:

$$\pi = e^{3/2} \prod_{n=2}^{\infty} e\left(1 - \frac{1}{n^2}\right)^{n^2}$$

π ~ PI(N) / (1 + 1/(2***N**) - 1/(8***N**^2)), and

π ~ ***PN(N)*** / (1 + 1/(2 * ***N***) - 1/(8 * ***N*2**) + ***13/(144 * N*3) - 77/(1152 * *N*4)***)

The challenge for the implementation here lies in the limited data format used by the HP-41. With just a 10-digit mantissa capability the iterative routines are likely to fail due to cumulative errors, thus we can forget about using FOCAL routines – at least not straightforward ones, anyway.

I decided to give MCODE a chance, to see if three more digits would make a difference – not expecting it to work but lo and behold it actually does a little good – albeit it can't cross the accuracy barrier we're up against, of course.

The function **PPIE** expects the number of terms to calculate in X, and returns the pi approximation already adjusted with the four corrections mentioned above. With the stated limitations it appears that the sweet spot appears for n=35 terms, giving a result with an absolute percent error of exactly zero compared to the native 10-digit value in the calculator.

The table below shows the logged details of the tests performed. Notice how things go south once the sweet spot is passed – due to the platform limitations. I have also included the execution time (on V41 with default settings, definitely not in TURBO mode)

n	result	|Delta%|	Time H:MMSS
5	3.141630979	1.2199E-05	0.000174
10	3.141593984	4.2335E-07	0.000297
15	3.141592834	5.7296E-08	0.000438
20	3.141592696	1.3369E-08	0.000568
25	3.141592666	3.8197E-09	0.000698
30	3.141592658	1.2732E-09	0.000829
35	3.141592654	0	0.00096
40	3.141592652	6.3662E-10	0.001088
45	3.141592651	9.5493E-10	0.001219
50	3.14159265	1.2732E-09	0.001337
55	3.141592648	1.9099E-09	0.001468
60	3.141592644	3.1831E-09	0.001606

And here's the MCODE listing with all the details of the implementation:

Header	AD5A	*085*	**"E"**	
Header	AD5B	*009*	**"I"**	
Header	AD5C	*010*	**"P"**	
Header	AD5D	*010*	**"P"**	*Ángel Martin*
PPIE	**AD5E**	*2A9*	*?NC XQ*	*Show "RUNNING" - leaves F8 as-is*
	AD5F	*13C*	*->4FAA*	*[RUNMSG]*
	AD60	2A0	SETDEC	
	AD61	*135*	*?NC XQ*	*naturalize the input*
	AD62	*134*	*->4D4D*	*[NATX4]*
	AD63	04E	C=0 ALL	
	AD64	35C	PT= 12	***C = 1***
	AD65	222	C=C+1 @PT	
	AD66	070	N=C ALL	initial N=1
	AD67	1A0	A=B=C=0	*zero trinity*
	AD68	*089*	*?NC XQ*	*current sum*
	AD69	*064*	*->1922*	*[STSCR]*
LOOPN	AD6A	3CC	?KEY	
	AD6B	360	?C RTN	
	AD6C	0B0	C=N ALL	*k-1*
	AD6D	*1E1*	*?NC XQ*	*{A,B} = C+1*
	AD6E	*100*	*->4078*	*[INCC10]*
	AD6F	070	N=C ALL	*k*
	AD70	*22D*	*?NC XQ*	*1/k*
	AD71	*060*	*->188B*	*[1/X_10]*
	AD72	*13D*	*?NC XQ*	*1/k^2*
	AD73	*060*	*->184F*	*[MP1_10]*
	AD74	2BE	C=-C-1 MS	*sign change*
	AD75	11E	A=C MS	*same in 13-digit form*
	AD76	*001*	*?NC XQ*	*1-1/k^2*
	AD77	*060*	*->1800*	*[ADDONE]*
	AD78	3C4	ST=0	
	AD79	*121*	*?NC XQ*	*Ln(1-1/k^2)*
	AD7A	*06C*	*->1B48*	*[LN13]*
	AD7B	0B0	C=N ALL	*k*
	AD7C	*13D*	*?NC XQ*	*k.Ln(1-1/k^2)*
	AD7D	*060*	*->184F*	*[MP1_10]*
	AD7E	0B0	C=N ALL	*k*
	AD7F	*13D*	*?NC XQ*	*k^2.Ln(1-1/k^2)*
	AD80	*060*	*->184F*	*[MP1_10]*
	AD81	*001*	*?NC XQ*	*1+k^2.Ln(1-1.k^2)*
	AD82	*060*	*->1800*	*[ADDONE]*
	AD83	*0D1*	*?NC XQ*	*current sum*
	AD84	*064*	*->1934*	*[RCSCR]*
	AD85	*031*	*?NC XQ*	*updated sum*
	AD86	*060*	*->180C*	*[AD2-13]*
	AD87	*089*	*?NC XQ*	*current sum*
	AD88	*064*	*->1922*	*[STSCR]*
	AD89	0B0	C=N ALL	*current term*
	AD8A	10E	A=C ALL	*put k in A for compares*
	AD8B	0F8	READ 3(X)	*number of terms*
	AD8C	36E	?A#C ALL	*all done?*
	AD8D	***2EF***	***JC -35d***	*do next*

	AD8E	0A9	?NC XQ	final product
	AD8F	064	->192A	[EXSCR] - {A,B} <-> {Q,+}
	AD90	04E	C=0 ALL	
	AD91	35C	PT= 12	**C = 1.5**
	AD92	050	LD@PT- 1	
	AD93	150	LD@PT- 5	
	AD94	025	?NC XQ	
	AD95	060	->1809	[AD1_10]
ADJUST	AD96	0AE	A<>C ALL	save product result:
	AD97	070	N=C ALL	13-digit sign & exp
	AD98	0CE	C=B ALL	
	AD99	128	WRIT 4(L)	13-digit mantissa
CT4	AD9A	0F8	READ 3(X)	n
	AD9B	10E	A=C ALL	
	AD9C	135	?NC XQ	n^2
	AD9D	060	->184D	[MP2_10]
	AD9E	13D	?NC XQ	n^4
	AD9F	060	->184F	[MP1_10]
	ADA0	04E	C=0 ALL	
	ADA1	35C	PT= 12	
	ADA2	050	LD@PT- 1	
	ADA3	050	LD@PT- 1	**c = 1152**
	ADA4	150	LD@PT- 5	
	ADA5	090	LD@PT- 2	
	ADA6	130	LDI S&X	
	ADA7	003	CON:	
	ADA8	13D	?NC XQ	1152.n^4
	ADA9	060	->184F	[MP1_10]
	ADAA	239	?NC XQ	1/1152.n^4
	ADAB	060	->188E	[ON/X13
	ADAC	04E	C=0 ALL	
	ADAD	2DC	PT= 13	
	ADAE	250	LD@PT- 9	
	ADAF	1D0	LD@PT- 7	**C = -77**
	ADB0	1D0	LD@PT- 7	
	ADB1	130	LDI S&X	
	ADB2	001	CON:	
	ADB3	13D	?NC XQ	-77/1152.n^4
	ADB4	060	->184F	[MP1_10]
	ADB5	089	?NC XQ	-77/1152.n^4
	ADB6	064	->1922	[STSCR]
CT3	ADB7	0F8	READ 3(X)	n
	ADB8	10E	A=C ALL	
	ADB9	135	?NC XQ	n^2
	ADBA	060	->184D	[MP2_10]
	ADBB	0F8	READ 3(X)	n
	ADBC	13D	?NC XQ	n^3
	ADBD	060	->184F	[MP1_10]
	ADBE	04E	C=0 ALL	
	ADBF	130	LDI S&X	
	ADC0	144	CON:	**C = 144**
	ADC1	07C	RCR 4	
	ADC2	130	LDI S&X	

	ADC3	002	CON:	
	ADC4	*13D*	*?NC XQ*	*144.n^3*
	ADC5	*060*	*->184F*	*[MP1_10]*
	ADC6	*239*	*?NC XQ*	*1/144.n^3*
	ADC7	*060*	*->188E*	*[ON/X13*
	ADC8	04E	C=0 ALL	
	ADC9	35C	PT= 12	
	ADCA	050	LD@PT- 1	***C = 13***
	ADCB	0D0	LD@PT- 3	
	ADCC	130	LDI S&X	
	ADCD	001	CON:	
	ADCE	*13D*	*?NC XQ*	*13/144.n^3*
	ADCF	*060*	*->184F*	*[MP1_10]*
	ADD0	*0D1*	*?NC XQ*	*-77/1152.n^4*
	ADD1	*064*	*->1934*	*[RCSCR]*
	ADD2	*031*	*?NC XQ*	*13/144.n^3 -77/1152.n^4*
	ADD3	*060*	*->180C*	*[AD2-13]*
	ADD4	*089*	*?NC XQ*	*13/144.n^3 -77/1152.n^4*
	ADD5	*064*	*->1922*	*[STSCR]*
CT2	ADD6	0F8	READ 3(X)	π ~ **PN(N)** / (1 + 1/(2***N**) - 1/(8*N^2))
	ADD7	10E	A=C ALL	
	ADD8	*135*	*?NC XQ*	*n^2*
	ADD9	*060*	*->184D*	*[MP2_10]*
	ADDA	04E	C=0 ALL	
	ADDB	130	LDI S&X	***c = -8***
	ADDC	098	CON:	
	ADDD	23C	RCR 2	
	ADDE	*13D*	*?NC XQ*	*-8.n^2*
	ADDF	*060*	*->184F*	*[MP1_10]*
	ADE0	*239*	*?NC XQ*	*-1/8.n^2*
	ADE1	*060*	*->188E*	*[ON/X13*
	ADE2	*0D1*	*?NC XQ*	*13/144.n^3 -77/1152.n^4*
	ADE3	*064*	*->1934*	*[RCSCR]*
	ADE4	*031*	*?NC XQ*	*-1/8.n^2+13/144.n^3 -77/1152.n^4*
	ADE5	*060*	*->180C*	*[AD2-13]*
	ADE6	*089*	*?NC XQ*	*-1/8.n^2+13/144.n^3 -77/1152.n^4*
	ADE7	*064*	*->1922*	*[STSCR]*
CT1	ADE8	0F8	READ 3(X)	*n*
	ADE9	10E	A=C ALL	
	ADEA	*01D*	*?NC XQ*	*2n*
	ADEB	*060*	*->1807*	*[AD2_10]*
	ADEC	*239*	*?NC XQ*	*1/2n*
	ADED	*060*	*->188E*	*[ON/X13*
	ADEE	*0D1*	*?NC XQ*	*-1/8.n^2+13/144.n^3 -77/1152.n^4*
	ADEF	*064*	*->1934*	*[RCSCR]*
	ADF0	*031*	*?NC XQ*	*1/2n -1/8.n^2+13/144.n^3 - 77/1152.n^4*
	ADF1	*060*	*->180C*	*[AD2-13]*
	ADF2	*001*	*?NC XQ*	*1+1/2n -1/8.n^2+13/144.n^3 - 77/1152.n^4*
	ADF3	*060*	*->1800*	*[ADDONE]*
	ADF4	*121*	*?NC XQ*	
	ADF5	*06C*	*->1B48*	*[LN13]*
	ADF6	2BE	C=-C-1 MS	*sign change*

ADF7	11E	A=C MS	*ditto for 13-digit form*
ADF8	0B0	C=N ALL	*recover product result:*
ADF9	158	M=C ALL	*13-digit sign & exp*
ADFA	138	READ 4(L)	*13-digit mantissa*
ADFB	*031*	*?NC XQ*	*Ln(PN(N))*
ADFC	*060*	*->180C*	*[AD2-13]*
ADFD	*035*	*?NC XQ*	*PN(N)*
ADFE	*068*	*->1A0D*	*[EXP13]*
ADFF	*331*	*?NC GO*	*Overflow, DropST, FillXL & Exit*
AE00	*002*	*->00CC*	*[NFRX]*

So here you have it, quite a long code but conceptually not a complicated one – such is the nature of the MCODE game sometimes.

PS.- Jean-François Garnier has provided the following FOCAL routine that cleverly overcomes the 10-digit accuracy issue to effectively reach good results with about 45 terms (that is 10 more than the MCODE version, using the first two correction factors instead of four - not bad at all!)

01	**LBL "PN2"**		21	+	
02	"RUNNING"		22	DSE 01	
03	AVIEW		23	GTO 00	*; sum endloop --^*
04	STO 00	; N	24	1.5	
05	E		25	+	
06	-		26	RCL 00	
07	STO 01	*; control loop 1..N-1*	27	2	
08	0		28	*	
09	LBL 00	*; sum loop <---*	29	1/X	
10	RCL 01		30	RCL 00	
11	E		31	X^2	
12	+	*;n=2..N*	32	8	
13	X^2		33	*	
14	ENTER^		34	1/X	
15	1/X		35	-	
16	CHS		36*	LN1+X	*; correction factor*
17	LN1+X		37	-	
18	*		38	E^X	
19	E		39	CLD	
20	+		40	END	

Appendix.- Integral Pie

And what about using an integral form, you may ask? Well, mixed results here to report. The good news is that using a simple simple expression like the one below works like a charm with a quick call to FROOT:

$$\pi = \int_{-1}^{1} \frac{dx}{\sqrt{1-x^2}}.$$

Setting FIX 9:

XEQ "PITG" => 3.141592654

1	LBL "PITG"
2	"*I"
3	0
4	ENTER^
5	1
6	FINTG
7	4
8	*
9	RTN
10	LBL "*I"
11	CHS
12	E
13	-
14	SQRT
15	END

References: See https://functions.wolfram.com/Constants/Pi/07/

So far so good, however I've not succeeded with other more complex derivations included in other "Short & Sweet Challenge" threads, such as those shown below:

$$\int_0^x \left(\frac{\sin t}{t} e^{t/\tan t}\right)^x dt - \frac{x^x}{\Gamma x} = 0$$

Which doesn't converge no matter how I try it, and:

$$\pi = \frac{1}{W_0(1)} \int_0^{\pi} \log\left(1 + \frac{\sin t}{t} e^{t \cot t}\right) dt.$$

Which includes pi in the definition of pi, if you see my circular point…

See the original thread for more details:

HP Challenge VA515 - 2021-03-14 - SRC 009 Pi Day 2021 Special.pdf

Salimin-Brent Algorithm.

In 1976 Eugene Salamin and Richard Brent independently discovered a new algorithm for pi, which is based on the arithmetic-geometric mean and some ideas originally due to Gauss in the 1800s (although for some reason Gauss never saw the connection to computing pi). This algorithm produces approximations that converge to pi much more rapidly than any classical formula. It may be stated as follows:

Set $a_0 = 1, b_0 = 1/\sqrt{2}$ and $s_0 = 1/2$. For $k = 1, 2, 3, \cdots$ compute

$$\begin{aligned} a_k &= \frac{a_{k-1} + b_{k-1}}{2} \\ b_k &= \sqrt{a_{k-1} b_{k-1}} \\ c_k &= a_k^2 - b_k^2 \\ s_k &= s_{k-1} - 2^k c_k \\ p_k &= \frac{2a_k^2}{s_k} \end{aligned}$$

$$\pi \approx \frac{4\, a_N^2}{1 - \sum_{k=1}^{N} 2^{k+1}(a_k^2 - g_k^2)}$$

Then pk converges quadratically to pi. This means that each iteration of the algorithm approximately doubles the number of correct digits of pi. To be specific, successive iterations produce 1, 4, 9, 20, 42, 85, 173, 347, and 697 correct digits of pi. However, each of these iterations must be performed using a level of numeric precision that is at least as high as that desired for the final result; and that unfortunately means just three iterations are meaningful for the HP-41's 10-digit precision ceiling.

The FOCAL routine below implements the algorithm for the PIE ROM:

1	LBL "SBPI"		23	CHS	
2	2		24	RCL M	*a(k)*
3	1/X	*1/2*	25	X^2	*a(k)^2*
4	STO O		26	+	*c(k)*
5	SQRT	*a(0)*	27	RCL Y	*k,003*
6	STO N		28	INT	*k*
7	E		29	2^X-1	*2^(k) - 1*
8	STO M	*b(0)*	30	E	
9	0,003		31	+	*2^k*
10	+	*1.003*	32	*	*2^k . C(k)*
11	LBL 00		33	CHS	
12	RCL M	*b(k-1)*	34	ST+ O	*s(k) in O*
13	RCL N	*a(k-1)*	35	RDN	*k,003*
14	+		36	ISG X	*do next?*
15	2		37	GTO 00	*yes*
16	/	*a(k)*	38	RCL M	*a(k)*
17	X<> M	*b(k-1)*	39	X^2	*a(k)^2*
18	RCL N	*a(k-1)*	40	ST+ X	*2(a(k)^2*
19	*		41	RCL O	*s(k))*
20	SQRT	*b(k)*	42	/	*p(k)*
21	STO N		43	END	
22	X^2	b(k)^2			

Heretical Pi (*an early April's 1st joke* :-)

Inspired by the clever elegance in the Salamin-Brent method I wondered whether a non-iterative form could be extrapolated from the same approach, using the same starting "anchor" points { 1, 1/sqr(2) } and based on the AGM and GHM means; plus *using a "magic" fudge factor* "k" to make it all somehow work out. A totally absurd anathema but just for fun, consider the following expression:

$$pi = \frac{2.\,agm^2}{\frac{1}{2} - (agm^2 - ghm^2).\,2^k}$$

One could even attempt to legitimize this derangement by stating that the fudge factor "k" is based on the Erdós-Borwein constant, ε_{EB} as follows: *(oh this is getting too weird, or is it?)*

$$\frac{5\,(\mathcal{E}_{EB}+2)}{\mathcal{E}_{EB}+16} \approx 1.024239673481$$

And this (see left) is the tonge-in-cheek, no-nonsensical (uh?) FOCAL routine used that consolidates the heresy and materializes this wondrous, innovative bluff.

Trying it out for size:

XEQ "MYPI" => 3.141592654

If you thought this made no sense (say what?) then wait to read my dissertation on the search - and finding - of a new transcendent number τ (a.k.a π's *cousin*) through which the length of the ellipse circumference can be expressed in a closed form by:

L = 2. τ. sqr(a^2+b^2)

Where a,b are, of course, the semi-axis of said ellipse.

Not convinced yet? Well, perhaps you may want to check my string-theory-based quick *proof of the Riemann hypothesis* in the next section of the manual…

Note: see here for another rant on the subject, it's worth reading – but keep your mind open!

1	LBL "MYPI"	
2	*2*	
3	SQRT	
4	1/X	
5	*E*	
6	AGM	*agm(1, 1/sqr(2))*
7	X^2	*agm^2*
8	STO 00	
9	*2*	
10	SQRT	
11	1/X	
12	*E*	
13	GHM	*ghm(1,1/sqr(2))*
14	X^2	*ghm^2*
15	-	*agm^2 - ghm^2*
16	*2*	
17	ENTER	
18	*1.024239678*	*k*
19	Y^X	*2^k*
20	CHS	*-2^k*
21	*	*-(agm^2 - ghm^2) /*
22	0.5	
23	+	*1/2- (agm^2-ghm^2*
24	1/X	
25	RCL 00	*agm^2*
26	ST+ X	*2.agm^2*
27	*	*final result*
28	END	*done*

Extra bonus: speaking of Erdós-Borwein, here's a nice MCODE Utility and corresponding FOCAL routine side by side to calculate this constant – using the definition series:

https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93Borwein_constant

$$E = \sum_{n=1}^{\infty} \frac{1}{2^n - 1} \approx 1.606695152415291763\ldots$$

```
01 LBL "EBC"
02 0
03 E
04 LBL 00
05 2^X-1
06 LASTX
07 X<>Y
08 1/X
09 ST+ Z
10 FS? 10
11 VIEW Z
12 X=0?
13 GTO 02
14 RDN
15 ISG X
16 NOP
17 GTO 00
18 LBL 02
19 X<> Z
20 CLD
21 END
```

Header	A6E4	082	"B"	Erdós-Borwein constant
Header	A6E5	005	"E"	Ángel Martin
EB	A6E6	18C	?FSET 11	
	A6E7	3B5	?C XQ	Stack lift
	A6E8	051	->14ED	[R_SUB]
	A6E9	1A0	A=B=C=0	zero trinity
	A6EA	070	N=C ALL	k=0
LOOP	A6EB	3CC	?KEY	converges in 30 iterations
	A6EC	360	?C RTN	
	A6ED	089	?NC XQ	current sum
	A6EE	064	->1922	[STSCR]
	A6EF	0B0	C=N ALL	n-1
	A6F0	1E1	?NC XQ	{A,B} = C+1
	A6F1	100	->4078	[INCC10]
	A6F2	070	N=C ALL	n
	A6F3	04E	C=0 ALL	
	A6F4	35C	PT=12	builds "2" in C
	A6F5	090	LD@PT- 2	
	A6F6	084	CLRF 5	Natural Ln
	A6F7	115	?NC XQ	
	A6F8	06C	->1B45	[LN10]
	A6F9	0B0	C=N ALL	
	A6FA	13D	?NC XQ	n.Ln(2)
	A6FB	060	->184F	[MP1_10]
	A6FC	048	SETF 4	substract one: e^x-1
	A6FD	035	?NC XQ	13-digit precision here
	A6FE	068	->1A0D	[EXP13]
	A6FF	239	?NC XQ	1/(2^x-1)
	A700	060	->188E	[ON/X13
	A701	0E8	WRIT 3(X)	n-th. term
	A702	351	?NC XQ	Check error tolerance
	A703	128	>4AD4	[TOLER4]
	A704	2FE	?C#0 MS	negative? (passes tolerance)
	A705	03F	JC +07	yes, exit loop
	A706	0A9	?NC XQ	
	A707	064	->192A	[EXSCR] - {A,B} <-> {Q,+}
	A708	0F8	READ 3(X)	
	A709	025	?NC XQ	new result(k)
	A70A	060	->1809	[AD1_10]
	A70B	303	JNC -32d	do next
EXIT	A70C	0A9	?NC XQ	
	A70D	064	->192A	[EXSCR] - {A,B} <-> {Q,+}
	A70E	0AE	A<>C ALL	Mant sign and exponent
	A70F	0DA	C=B M	Mantissa value
	A710	0E8	WRIT 3(X)	
	A711	3C1	?NC GO	Normal Function Return
	A712	002	->00F0	[NFRPU]

Wallis-based Approximations

Also included in the module are a handful of routines based on the infamous Wallis product expression for the approximation It's well known that said expression requires a very large number of terms to get a decent accuracy in the result, hence its usage is limited from a practical point of view. However, there are ways to go around that deficiency using "correction" factors or other modifications on top of the basic one.

Function	Description	Input	Author
WALLIS	Wallis formula (n in X)	n in X	*Ángel Martin*
"WP42	Wallis product Formula	n in X	*Gerson W. Barbosa*
"WPI	Wallis product Formula	n in X	*Jean-Marc Baillard*
"WPIH	Wallis Formula w/ Hyperbolics	n in X	*Werner*
"CFWP	Conti. Fractions correction	n in X	*Martin-Barbosa*
"WWPI	Wallis-Wasicki Formula	n in X	*Gerson W. Barbosa*

- **WALLIS** is the MCODE implementation of the infamous Walls product. It requires a number of terms input in X and returns the estimation of pi to the stack X register.

$$\pi \approx 2\left(\frac{4}{3} \times \frac{16}{15} \times \frac{36}{35} \times \frac{64}{63} \times \cdots \times \frac{4n^2}{4n^2-1}\right)$$

The table below shows (left column) the results for different number of terms; note how the values get closer to the actual pi value when the Wallis formula is combined with a correction factor (right column), as we'll see next:

# of terms	Wallis Result	Wallis-Wasicki Result
10	3,067703 8100	3.14252 3109
100	3,1337 87496	3.14 1602424
1,000	3,14080 7792	3.14 1592798
10,000	3,14 15 14648	3.14 1593184
100,000	3,14 15626 18	n/a

example:

10,000 , XEQ "WALLIS" => 3,14 15 14648

Not much to write home about, to say the least, so let's see other more efficient approaches (read: fewer number of terms) while still based on the basic Wallis formula.

The two programs below are different versions contributed by forum members to compute the Wallis product (without correction factors). On the left using data registers and the RCL math (taken from an HP-42 solution); on the righ two more concise routines using only the stack.

01▸	LBL "W42"	42	X<>Y
02	STO 01	43▸	LBL 00
03	NOT	44	STO+ ST T
04	2	45	X<>Y
05	MOD	46	R↑
06	ENTER	47	RCL× ST T
07	STO+ ST X	48	RCL 03
08	E	49	X<>Y
09	-	50	SIGN
10	4	51	+/-
11	RCL× 01	52	STO× ST Z
12	E	53	X<> ST L
13	RCL- ST T	54	÷
14	×	55	X<> ST Z
15	R↑	56	RCL 01
16	STO+ ST X	57	STO+ ST X
17	+	58	X↑2
18	3	59	STO× 02
19	RCL× ST T	60	DSE ST X
20	-2	61	STO÷ 02
21	STO 02	62	R↓
22	RCL+ 01	63	RCL ST Y
23	X<>Y	64	SIGN
24	+	65	X<> ST Z
25	STO 03	66	ABS
26	RCL 02	67	X<> 04
27	X<> ST L	68	+/-
28	STO+ ST X	69	STO+ 03
29	RCL+ ST L	70	NOT
30	STO 04	71	+/-
31	-	72	NOT
32	RCL- 02	73	+/-
33	RCL× ST Z	74	X<> 04
34	+/-	75	DSE 01
35	4	76	GTO 00
36	RCL× 01	77	SIGN
37	RCL- ST Z	78	RCL+ ST T
38	RCL- 02	79	RCL× 02
39	STO÷ ST Y	80	ABS
40	X<>Y	81	END
41	R↓		

1	LBL "WPI"	*JM Baillard*
2	*2*	
3	LBL 01	
4	RCL Y	
5	ST+ X	
6	X^2	
7	ST* Y	
8	DSE X	
9	*/*	
10	DSE Y	
11	*GTO 01*	
12	RTN	
13	LBL "WPIH"	*Werner*
14	*2*	
15	LBL 02	
16	RCL Y	
17	ST+ X	
18	HACOS	
19	HTAN	
20	X^2	
21	*/*	
22	DSE Y	
23	*GTO 02*	
24	END	

Wallis-Wasicki formula.

See: https://www.hpmuseum.org/forum/post-139434.html#pid139434
See also: https://www.hpmuseum.org/forum/post-9194.html#pid9194

Gerson W. Barbosa has proposed a correction factor on top of the Wallis product for slightly more accurate results and definitely better efficiency. The correction factor is the finite continued fraction shown below, with a constant B(n) term pattern reflecting the number of terms used in the Wallis part of the combined formula.

$$2+\cfrac{4}{8n+3+\cfrac{3}{8n+4+\cfrac{15}{8n+4+\cfrac{35}{8n+4+\cfrac{63}{\ddots\,\cdots\,\cfrac{}{8n+4+\cfrac{4n^2-1}{8n+4}}}}}}}$$

So right off the shoe we could use the Continued Fractions engine to calculate the correction factor, which should definitely converge relatively quick given the large values for both A(n) and B(n). This is what the routine CWPI does, listed below:

1	LBL "CWPI"		23	RTN	
2	"*WP"		24	LBL 01	
3	2		25	ENTER^	
4	ENTER^		26	X^2	(n-1)^2
5	CF2V		27	4	
6	RCL 10		28	*	4(n-1)
7	WALLIS		29	ENTER^	
8	2		30	-	4(n-1)^2 - 1
9	/		31	X<>Y	n-1
10	*		32	8	
11	RTN		33	*	8(n-1)
12	LBL "*WP		34	4	
13	RCL 02		35	+	8(n-1)+4
14	ENTER^		36	/	A(n) = (4)n-1)^2
15	-	(n-1)	37	LBL 02	- 1)/[8(n-1)+4]
16	X#0?		38	RCL 10	N
17	GTO 01		39	8	
18	XEQ 02		40	*	8N
19	X<>Y		41	4	
20	ENTER^		42	+	B(n) = 8N+4
21	-	B(1)= 8N+3	43	X<>Y	
22	4	A(1)= 4	44	END	

The other approach is obviously to combine both the Wallis product and the correction factor at the same time, during the execution of the main body code segment. This is done in routine WWPI listed below:

01	LBL "WWPI"	16	X<> Z
02	4	17	ST/ Y
03	0	18	X<> L
04	8	19	R↓
05	RC* T	20	X<>Y
06	RC+ Z	21	DSE T
07	LBL 00	22	GTO 00
08	R↑	23	DSE X
09	RC+ X	24	+
10	ST* X	25	1/X
11	ST* T	26	0.5
12	DSE X	27	+
13	ST÷ T	28	x
14	X<>Y	29	END
15	ST+ Z		

Table of results/-

Uncorrected Wallis:

N	WP42	WPI	WPIH
10	3.067703807	3.067703807	3.067703807
100	3.133787499	3.133787499	3.133787499
1,000	3.140807756	3.140807756	3.140807756
10,000	3.141514015	3.141514015	3.141514015
100,000	3.141571397	3.141571397	3.141571397

The three versions are totally identical for any number of iterations.

Corrected Wallis:

n	WWPI	CWPI
10	3.141592654	3.142523109
100	3.141592666	3.141602424
1,000	3.141592682	3.141592798
10,000	3.141592768	3.141593184

The sweet spot appears to be n=1,000 for both, no doubt the workings of the finite continued functions term.

Pi/e using Continued Fractions

There are many different expressions related to pi and e using continued fractions, both with and without a clear pattern to the coefficients. As usual, some of them converge very slowly and aren't practical for the calculations - thus only have an academic value.

Amongst those useful for our purposes, we find these two for pi:

Routine name: **CFPI**

$$\pi = \cfrac{4}{1 + \cfrac{1^2}{3 + \cfrac{2^2}{5 + \cfrac{3^2}{7 + \ddots}}}}$$

Routine name: **CFP1**

$$\pi = 3 + \cfrac{1^2}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6 + \ddots}}}$$

With the following recurrent pattern parameters on each case being:

B(0) = 0
A(1) = 4 ; B(1) = 1
A(n) = (n-1)^2 ; B(n) = 2n-1

B(0)=3
A(n) = (2n-1)^2 ; B(n)= 6

And this one for e, beautifully simple and even more efficient for the calculation:

$$e = 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{2}{3 + \cfrac{3}{\ddots}}}}$$

with the following recurrent parameters:

B(0)=2 ;
A(1) = 1 ; B(1)= 1
A(n)=(n-1) ; B(n)=n

XEQ "CFE" => 2.718281830 ; with just 5 terms needed

XEQ "CFPI" => 3.141592640 ; with 420 terms needed.

XEQ "CFP1" => 3.141592652 ; with 14 terms needed

As always, you can set flag 10 to see the progress of the convergence in the display.

References: *https://mathworld.wolfram.com/eContinuedFraction.html*
https://en.wikipedia.org/wiki/Continued_fraction

The Path not taken:-

Two of the non-practical continued fractions are shown below, for the $\pi/2$ and $4/\pi$ cases– both requiring many thousands of iterations to achieve decent accuracy (say 5 decimal digits or better), and thus taking an awfully long execution time even on V41 in turbo mode.

$$\frac{\pi}{2} = 1 - \cfrac{1}{3 - \cfrac{2\cdot 3}{1 - \cfrac{1\cdot 2}{3 - \cfrac{4\cdot 5}{1 - \cfrac{3\cdot 4}{3 - \cfrac{6\cdot 7}{1 - \cfrac{5\cdot 6}{3 - \dots}}}}}}}$$

$$\frac{4}{\pi} = 1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{2 + \dots}}}}$$

Brouncker's formula:

Programmed as follows:

1	LBL "CFPI2"		32	RCL X	
2	LBL A		33	E	
3	"$P"		34	-	*(n-2)*
4	E	*B(0)= 1*	35	*	*(n-1).(n-2)*
5	ENTER^		36	CHS	*A(2n+1)= -(n-1).(n-2)*
6	CF2V	*π/2*	37	3	*B(2n+1)= 3*
7	ST+ X	*π*	38	X<>Y	
8	RTN		39	RTN	
9	GTO A		40	LBL "CFPI4"	
10	LBL "$P"		41	LBL B	
11	FS? 10		42	"*P"	
12	VIEW 00		43	E	*B(0) = 1*
13	3	*B(1)= 3*	44	ENTER^	
14	RCL 02	*n*	45	CF2V	*4/π*
15	X=1?		46	1/X	*π/4*
16	CHS	*A(1)= -1*	47	4	
17	X<0?		48	*	*π*
18	RTN		49	RTN	
19	ODD?	*odd?*	50	GTO B	
20	GTO 01	*yes, divert*	51	LBL "*P"	
21	RCL 02	*n*	52	FS? 10	
22	E		53	VIEW 00	
23	+	*(n+1)*	54	RCL 02	*n*
24	*	*n.(n+1)*	55	ST+ X	*2n*
25	CHS	*A(2n)= -n.(n+1)*	56	E	
26	E	*B(2n)= 1*	57	-	*2n-1*
27	X<>Y		58	X^2	*A(n) = (2n-1)^2*
28	RTN		59	2	*B(n)= 2*
29	LBL 01	*odd term,n#1*	60	X<>Y	
30	E		61	END	
31	-	*(n-1)*			

Random Pie – Monte Carlo method

This section uses a variation of the Monte Carlo strategy to evaluate both pi and e. It's not, however, based in circle relationships derived from randomly throwing needles or shooting at targets, but on probability theory instead. It was explained by Valentín himself in his [HP Challenge VA511 - 2020-03-14 - SRC 006 Pi Day 2020 Special.pdf]

Quoting directly from that article:

"It's quite simple, actually. My recent program is this:

```
1  DESTROY ALL @ RANDOMIZE 1 @ FOR K=1 TO 5 @ N=10^K @ S=0
2  FOR I=1 TO N @ IF NOT MOD(IROUND(RND/RND),2) THEN S=S+1
3  NEXT I @ P=S/N @ STD @ DISP N, @ FIX 3 @ DISP 5-P*4 @ NEXT K
```

which is computing the probability that the closest integer to A/B is even, where A and B are uniformly distributed random numbers in [0,1), as produced by the RND keyword. Each time the rounded value is even (i.e., it's 0 modulo 2) the number of favorable outcomes (S) is incremented by one (see line 2). After N tries have been sampled, the probability P for the even case will be the number of favorables outcomes (S) divided by the number of tries (N), thus we have the estimated probability P = S/N.
*But I know from theory that in the limit, for N -> Infinity, the exact probability P = (5-Pi)/4, so isolating Pi we have Pi = 5-P*4, which is displayed by the program in line 3 above."*

Note that he goes on to include yet another possible approach, which results in an even shorter BASIC program. Here's the explanation:

"Now, my earlier program, the one-liner, namely:

```
10 INPUT K @ N=0 @ FOR I=1 TO K @ N=N-MOD(IROUND(RND/RND),2) @ NEXT I @ DISP 1-4*N/K
```

is computing the probability that the closest integer to A/B is odd, where A and B are uniformly distributed random numbers in [0,1), as produced by the RND keyword. Each time the rounded value is odd (i.e., isn't 0 modulo 2) the number of favorable outcomes (N) is decremented by one, and after K tries have been sampled, the probability for the odd case will be the number of favorable outcomes (-N) divided by the number of tries (K), thus we have the estimated probability P = -N/K.

*As the probability of the rounded division being either even or odd is 1 (certainty), the probability for the odd case is 1 minus the probability for the even case, thus it's P = 1-(5-Pi)/4 = (Pi-1)/4, so isolating Pi we have Pi = 1+4*P = 1+4*(-N/K) = 1-4*N/K, which is then displayed by the one-line program."*

I chose to use the first approach in this module, partially because it also requires the IROUND function, and I was intrigued by it. I ended up writing a short MCODE utility for that purpose, which facilitates the porting of the BASIC code to HP-41 FOCAL, shown in next page.

With regard to the e calculation, the source has also been Valentín's *HP Challenge VA030 - Short Sweet Math Challenge 25 San Valentin Special - Weird Math.pdf.* In that thread there's one section (the first "concoction") about calculating a "weird limit" that can be used for the calculation of e (making the sum--to-exceed s=1).

"The limit average count for the sum of a series of [0,1) uniformly distributed random numbers to exceed 1 is exactly e = 2.71828182845904523536+, the base of the natural logarithms, which is pretty "weird" and can be considered an analog of Buffon's Needle experiment to estimate the value of Pi. Here we don't throw needles on a grid but merrily add up random numbers keeping count and we get e instead."

"This is the general formula to numerically compute the theoretically exact value and my simple 1-line, 53-byte HP-71B program to instantly compute them given the sum to exceed: "

$$f(x) = \sum_{k=0}^{[x]} (-1)^k \frac{(x-k)^k}{k!} e^{x-k}$$

```
1 DESTROY ALL @ INPUT X @ S=0 @ FOR K=0 TO IP(X) @ S=S+(K-X)^K/FACT(K)*EXP(X-K) @
NEXT K @ DISP S
```

For the porting we'll certainly need the new **IROUND** utility and obviously capable random number capabilities, which shouldn't be much of a problem using the SandMath's functions **SEEDT** and **RNDM**. E'll use a time-generated initial seed (input zero for SEEDT), and RNDM will do the work using the well-known RNG recurrence:

$$r(k+1) = FRC\,[\,r(k) * 9{,}821 + 0.211327\,]$$

A few results are given in the table below:

Iterations	MCE	MCPI
10	2.800000000	3.000000000
100	2.850000000	3.120000000
1,000	2.705000000	3.136000000
10,000	2.717400000	3.131600000
100,000	2.717760000	3.149320000
1,000,000		

As you can see from the table results both routines require a very large number of iterations to get to a reasonably accurate result, which of course was expected as "it 'comes with the territory" when resorting to this type of approaches. See below for the actual program code.

1	LBL "MCE"		10	LBL "MCPI"	
2	**LBL A**		11	**LBL B**	
3	STO 01	*number of iterations*	12	STO 00	*number of iterations*
2	*E*	*sum limit*	11	*0*	*initial value*
3	*0*		12	**SEEDT**	*Time-based Seed*
4	STO 00	*initial count*	13	**LBL 11**	
3	E^X		12	**RNDM**	*PPC Method +*
4	**SEEDT**	*initial seed*	13	**RNDM**	*PPC Method +*
5	LBL 01		14	/	
4	CLX	*reset sum*	13	**IROUND**	
5	LBL 00		14	*2*	
6	ISG 00	*increase count*	15	MOD	
5	NOP		14	-	
6	**RNDM**	*PPC Method +*	15	FS? 10	
7	+	*update sum*	16	VIEW Y	
6	FS? 10	*need to show?*	16	DSE Y	
7	VIEW Z	*yes, oblige*	17	GTO 11	
8	X<Y?	*sum less than limit?*	17	RCL 00	*number of iterations*
7	GTO 00	*yes, get next RAN*	18	/	
8	DSE Z	*decrease counter*	18	*-4*	
9	GTO 01	*do next if not finished*	19	*	
8	RCL 00	*final count*	19	*E*	
9	RCL 01	*number of iterations*	20	+	
10	/		20	CLD	
9	CLD		21	RTN	
10	RTN		21	GTO B	
11	GTO A		22	END	

Note:- The poor-man version of **IROUND** would consist of setting FIX 0 before the LBL 11 loop, and adding an INT instruction after the division of both random numbers (i.e. replacing IROUND with INT). That's almost equivalent but doesn't handle the EVEN condition for the result, i.e. IROUND(5.5)=5 whereas INT(4.5) in FIX 0 is equal to 4 instead. Not a show-stopper though, considering how unlikely it is to find such an occurrence amongst the hundreds of random points used by the routine.

Humble Pie – Series Correction, "Speed it up!"

Yet another wonderful contribution by Mr. Albillo's at the top of his game - taken from the challenge thread [HP Challenge VA125 - 2006-07-12 - HP-15C Mini-challenge Speeding it up.pdf]

Here's the direct description from that thread, read on and enjoy !

"As stated in the challenge's description, the task is to find a way to use the well-known Gregory-Leibnitz series to compute Pi to 10 correct places while keeping program size and running time small.

$$\pi = 4\sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1} = 4\left(\frac{1}{1}-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+-\cdots\right)$$

A direct approach seems doomed to failure as this series converges so incredibly slowly that millions of terms must be added up to get no more than 6 or 7 correct digits, let alone 10. To clearly demonstrate it, this simple 15- step HP-15C program, which will serve as the basis for my solutions, will add up any specified even number of terms from the series:

```
01*LBL A     06 0         11 STO 0
02 STO I     07*LBL 0     12 RCL/I
03 STO+I     08 DSE I     13 +
04 4         09 RCL 0     14 DSE I
05 STO O     10 CHS       15 GTO 0
```

To improve accuracy, the program begins adding up the smallest terms and goes backwards until it reaches the largest term, 1. Upon running it, you'll see that, as expected, the convergence is awfully slow. Let's try to add 4 terms, then 44, then 444:

```
4 , GSB A -> 2.895238096 (barely one correct digit)
44 , GSB A -> 3.118868314 (barely three correct digits)
444, GSB A -> 3.139340404 (barely four correct digits)
```

This last result took almost 7 minutes, yet we've got no more than four not-so-correct digits, so the situation seems hopeless. At this point, it seems we can do no better than try some relatively complicated techniques, such as the Euler-McLaurin formula or extrapolation mechanisms for summation of infinite, alternating series such as this one. This would incur in a serious penalty in vastly increased complexity and program size, as seen in several working programs posted by contributors.

A bit of sleuthing:

However, math is full of surprises and serendipitous findings are bound to happen when and where you least expect them, as we'll immediately see.

Let's use our basic program to add up exactly 50 terms:

```
50, GSB A -> 3.121594653
```

Now, this has a fairly large error, as we're getting 3.12+ instead of 3.14+, so that the 3rd digit is already 2 units off. But, don't you notice something truly eerie ? Yes, we get a "2" where a "4" should be. But the following three digits (159) are correct ! Then we get another wrong digit, a "4" which should be a "2", but then the next three digits (653) are once again correct !!

Let's align our value and the correct Pi value and carefully examine the differences:

```
Sum -> 3.121594653
PI  -> 3.141592653 (58979...)
       ----------------------
       +2     -2
```

which, in absolute values means:

```
+0.02 -0.000002
```

Let's see if this is just a weird coincidence, or else it also happens for other numbers of terms being added up. Let's try 100 terms, for instance:

```
100, GSB A -> 3.131592904
              3.141592654
              ------------------
                +1   -25
             +0.01  -0.00000025
```

and we see that our initial impression does hold, because after one wrong digit, the subsequent four digits (1592) are indeed correct, then another a couple of wrong digits, and once again another correct digit follows.

Let's call these two corrections' C1 and C2 (i.e: +0.02 and -0.000002 for 50 terms, +0.01 and -0.00000025 for 100 terms, respectively) and see how they relate to the number of terms being used. A little insight or a little data-fitting will allow us to issue the following plausible, tentative hypothesis, where N is the number of terms:

```
C1 = 1/N
C2 = -0.25/N3 = -1/4N3
```

which do indeed work for N = 50 and N = 100 terms. Now we'll put our hypothesis to the test, by using it to predict the values of C1 and C2 for N=200 terms:

```
Prediction for N = 200 -> C1 = 1/200 = 0.005
C2 = -1/(4*2003) = -0.000000031
```

and we'll now check if they agree with actual results, by running our basic program with 200 as the input value:

```
200, GSB A -> 3.136592685
              3.141592654
              -----------------
                 +5   -31
```

which indeed do exactly agree with our predicted corrections, +0.005 and -0.000000031. At this point, we can be fairly sure that our empirical finding holds, and can then proceed to make use of it by simply computing one or both correction terms, C1 and C2, and using them to refine the sum provided by our basic program, as follows:

First version, using just one correction term, C1 = 1/N:

Just two little changes to our basic program will compute and add the correction term C1, resulting in a program just a single step longer, at 16 steps, yet much faster and accurate:

```
01*LBL A
02 STO I
03 STO+I      50,    GSB A -> 3.141594653 in 55"
04 1/X
05 4          error = 2E-6, actually all digits are correct except the "4"
06 STO O
07 X<>Y       100, GSB A -> 3.141592904 in 1'50"
08*LBL 0
09 DSE I      error = 2.5E-7
10 RCL 0
11 CHS        400, GSB A -> 3.141592658 in 7'45"
12 STO 0
13 RCL/I      error = 4E-9
14 +
15 DSE I
16 GTO 0
```

so this simple version, with just the one correction term C1 does achieve a 10-digit correct value (within 4 ulps) while using just 400 terms, in less than 8 minutes. That's many orders of magnitude better than the basic program could achieve, but we can do still much better:

Second version, using two correction terms, C1=1/N and C2=-1/4N^3:

A few stack manipulations will allow us to compute and use both correction terms, C1 and C2 while using just 5 additional steps, for a very small total of just 21 steps:

```
01*LBL A
02 STO I
03 STO+I      40, GSB A -> 3.141592651 in 40" (error = 3E-9)
04 1/X
05 ENTER
```

```
06 ENTER      50, GSB A -> 3.141592653 in 50" (error = 1E-9)
07 3
08 Y^X
09 4          62, GSB A -> 3.141592654 in 60" (error = 0)
10 STO O
11 /
12 -
13*LBL 0
14 DSE I
15 RCL 0
16 CHS
17 STO 0
18 RCL/I
19 +
20 DSE I
21 GTO 0
```

so this improved version needs to add up just 62 terms to return a full 10 correct-digit value within 60 seconds. Here's a table summarizing the different degrees of approximation using 0, 1, and 2 correction terms, for up to 60 terms added up:

N	bare series	+C1	+C1+C2	t
10	3.041839619	3.141839619	3.141589619	10"
20	3.091623807	3.141623807	3.141592557	20"
30	3.108268567	3.141601900	3.141592641	30"
40	3.116596557	3.141596557	3.141592651	40"
50	3.121594653	3.141594653	3.141592653	50"
60	3.124927144	3.141593811	3.141592653	60"

Further empirical confirmation:

As we've been able to indeed get 10 correct digits by using our empirically discovered corrections, we can be fairly confident that they are no mere coincidences but hold for greater number of terms added up and thus greater precision. To test this, just out of curiosity, these are the results for N = 500, 5000, 50000, 500000, and 5 million terms added up:

```
N = 500 terms added up
3.13959265558978323858464...
3.14159265358979323846264...
   +2     -2    +10  -122

N = 5,000 terms added up
3.14139265359179323836264339547950...
3.14159265358979323846264338327950...
    +2       -2      +10     -122

N = 50,000 terms added up
3.14157265358979523846264238327950410419716...
3.14159265358979323846264338327950288419716...
     +2         -2        +10       -122
```

```
N = 500,000 terms added up
3.14159065358979324046264338326950288419729139937510...
3.14159265358979323846264338327950288419716939937510...
      +2          -2          +10         -122

N = 5,000,000 terms added up
3.1415924535897932384646433832795027841971693993873058209749 4...
3.1415926535897932384626433832795028841971693993751058209749 4...
       +2             -2            +10            -122
```

Notice in particular the values for N = 5,000,000 terms: the 7th decimal is already in error by 2 units. But the next 13 digits are all correct ! Then, the following digit is also 2 units wrong. But the next 12 digits are again correct !! All in all, among the first 47 digits, only 3 of them are a few units wrong !

In other words, the original series converges incredibly slowly, granted, but the errors when you stop at N terms are extremely predictable and easy to compute, so you can increase your accuracy 3-fold or 5-fold by using just one or two easily derived correction terms.

Final notes

This empirical discovery, once made, can be substantiated by theory, and a nifty expression is arrived at which results in an asymptotic approximation to Pi based on the sum of the original series truncated to N terms plus a 'correction' series (the asymptotic component) in negative powers of N (1/N, $1/N^3$, etc) where the so-called Euler numbers are the coefficients.

Similar phenomena occur for constants other than Pi, for example similarly truncating the series:

```
Ln(2) = 1 - 1/2 + 1/3 - 1/4 + 1/5 - ...
```

results in:

```
Sum =   0.69314708055995530941723212125817656807551613436025525...
Ln(2) = 0.69314718055994530941723212145817656807550013436025525...
         1     -1             2           -16
```

and another asymptotic series can be theoretically substantiated, the required coefficients being now the so called "tangent numbers" instead: 1, -1, 2, -16, ...

Thanks for your interest and many excellent posted contributions, hope you enjoyed yourselves while working them out."

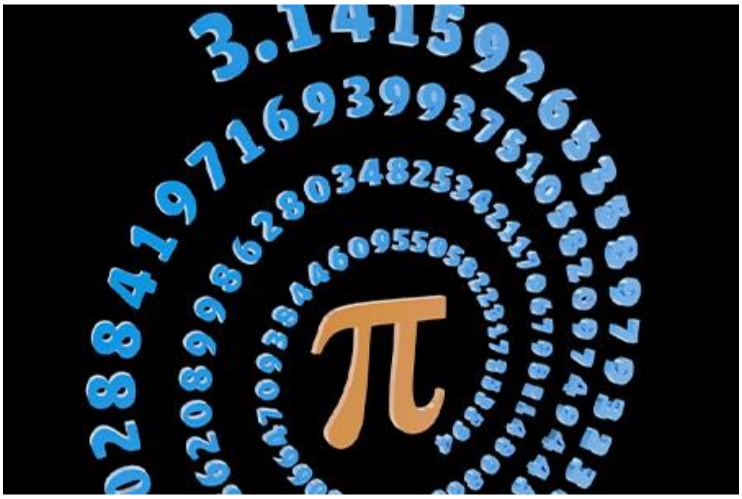
And here's how all this is applied to the HP-41 in this module, a deceptively simple code that however encompasses the devilish wizardry so well explained in the previous pages:

The routine is deservedly labeled "**VAPI**", I'm sure you'll understand why.

1	LBL "VAPI"	
2	STO M	
3	ST+ M	
4	1/X	
5	ENTER^	
6	X^3	
7	4	
8	STO N	
9	/	
10	LBL 00	
11	DSE M	
12	RCL N	
13	CHS	
14	STO N	
15	RC/ N	
16	+	
17	DSE M	
18	*GTO 00*	
19	END	

The table of results is shown below. Note the small number of iterations needed for a good accuracy, proof of the very efficient algorithm used.

N in X	Result
2	3.135416667
4	3.141331846
6	3.141555436
8	3.141583536
10	3.141589619
12	3.141591424
14	3.141592082
16	3.141592359
18	3.141592490
20	3.141592557



This concludes the first part of the manual. In the next section you'll find a short description of the MCODE and FOCAL programs to calculate many digits of pi and e.

Many Digits of Pi. *(by Peter Platzer, MoHPC Forum)*

https://www.hpmuseum.org/cgi-sys/cgiwrap...587#147587

The module includes the remarkable and impressive MCODE implementation of the Spigot algorithm by Peter Platzer, published in the Museum of HP Calculators forum. His description is available in the appendix, but here are the highlights:

The code asks for three inputs: The page where the MLDL ram starts to use, the number of digits and the base b to use (max = 5 for 5 digits at a time). One can set Flag 0 and the calc will stop at each group of digits and wait for a key to be pressed, otherwise it just keeps calculating …

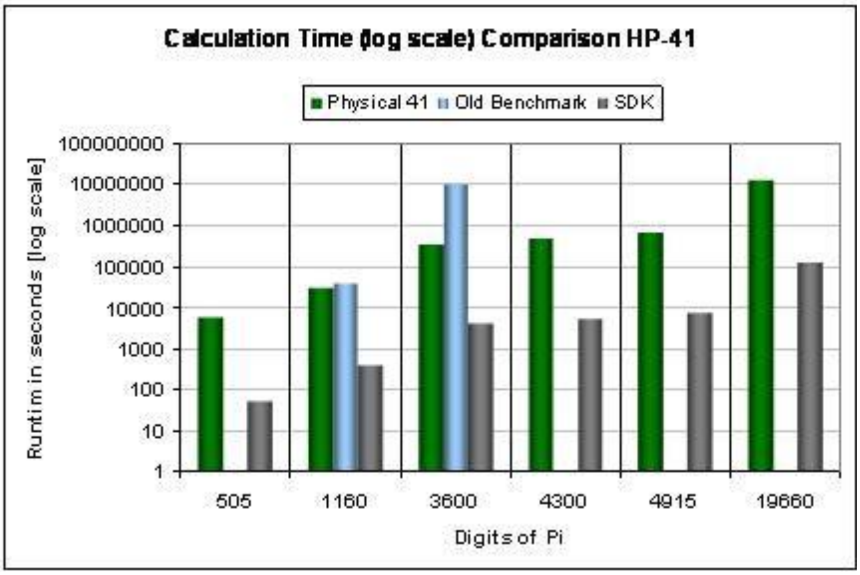
Setting Flag 1 will store the found digits in the same compressed format – each group of up to 5 digits is stored in 2 words, with the right nibble converted to hex. They are stored in reversed order though

In manual execution the function prompts for the number of digits to calculate (limited to 1999 by the prompt) and the destination page where to store them. This needs to be a q-RAM page to allow writes into it. The maximum number of digits is 4095 – which will fill up the page in its entirety.

The screens below show an example to calculate 1,046 digits to be stored in page B:

....

In an unmodified HP-41 it delivers 1,160 digits in about 9 hours 3,600 digits in about 4 days , and 4,915 digits in about 8 days. The chart below shows a comparison with the previous record-holding approaches described in the article.



```
; Many Digits of PI
; Spigot algorithm from Pi-book
; uses base b <= 5 to show 5 digits at a time
;Flag0 - wait for key press after each group is shown
;Flag1 - store result digits in reverse order from end (iStart)

;Input:
; Z : p - page number of start of MLDL ram to use
; Y : n - number of digits wanted
; X : base b in powers of 10
;--------------
; All Stack and Alpha is used for temp storage
; 3(X): i in dec, 1step 5(M): orig iStart in hex and 2 step
; 2(Y): tmp 6(N): last addr in hex and 2 step
; 1(Z): iBits in dec, 1 step 7(O): iBits in hex, 2 step
; 4(L): iStart in dec, 1 step 8(P): b|iStart in hex and 2 step
; 9(Q): q - remainder 0(T): page number in hex in C:[0]
;-------------
; All numbers are integers without exponent starting at C[0]
; User-Flag 0 -> wait for key press after each numbers shown. Stored in M-Flag 9
```

Extended precision: Pi to 1,000 places. *(by Ron Knapp, PPCCJ V8N6 p69)*

"*Compute the first 1,000 decimal digits of Pi in less than 11 hours, 30 minutes*". That was the friendly challenge put out by the PPC 'Journal", especially to members of the TI Personal Calculator Club, approximately a year ago. This challenge was repeated in the "Calcu-letter" of Popular Science Magazine, July 1981.

Up to the present time, I have heard of no serious attempts to eclipse this record. So,-- I decided to improve my own program. The program listed below computes Pi to 1,000 decimal places in just 8 hours, 30 minutes.

Ed. note: with 2x machines, and some will run Faster, (fastest reported so far was Emett Ingram (17) at 2.8x) a 4 hour, 1,000 digit Pi program is the state of the PPC art. How long will it be before someone places 100,000 digits of Pi on a cassette? A printer on the HP-IL would take nearly 45 minutes to print it on 70 feet of paper at 20 digits per line, 2 lines per second.

The first 1.000 decimal places of Pi contains 93 0s, 116 1s, 103 2s, 102 3s, 93 4s, 97 5s, 94 6s, 95 7s, 101 8s, and 106 9s. Below is "3 dot" followed by the first 1,000 decimals of Pi.

3 . 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8 4 6 2 6 4 3 3 8 3 2 7 9 5 0 2 8 8 4 1 9 7 1 6 9 3 9 9 3 7 5 1 0 5 8 2
0 9 7 4 9 4 4 5 9 2 3 0 7 8 1 6 4 0 6 2 8 6 2 0 8 9 9 8 6 2 8 0 3 4 8 2 5 3 4 2 1 1 7 0 6 7 9 8 2 1 4 8 0 8 6
5 1 3 2 8 2 3 0 6 6 4 7 0 9 3 8 4 4 6 0 9 5 5 0 5 8 2 2 3 1 7 2 5 3 5 9 4 0 8 1 2 8 4 8 1 1 1 7 4 5 0 2 8 4 1
0 2 7 0 1 9 3 8 5 2 1 1 0 5 5 5 9 6 4 4 6 2 2 9 4 8 9 5 4 9 3 0 3 8 1 9 6 4 4 2 8 8 1 0 9 7 5 6 6 5 9 3 3 4 4
6 1 2 8 4 7 5 6 4 8 2 3 3 7 8 6 7 8 3 1 6 5 2 7 1 2 0 1 9 0 9 1 4 5 6 4 8 5 6 6 9 2 3 4 6 0 3 4 8 6 1 0 4 5 4
3 2 6 6 4 8 2 1 3 3 9 3 6 0 7 2 6 0 2 4 9 1 4 1 2 7 3 7 2 4 5 8 7 0 0 6 6 0 6 3 1 5 5 8 8 1 7 4 8 8 1 5 2 0 9
2 0 9 6 2 8 2 9 2 5 4 0 9 1 7 1 5 3 6 4 3 6 7 8 9 2 5 9 0 3 6 0 0 1 1 3 3 0 5 3 0 5 4 8 8 2 0 4 6 6 5 2 1 3 8
4 1 4 6 9 5 1 9 4 1 5 1 1 6 0 9 4 3 3 0 5 7 2 7 0 3 6 5 7 5 9 5 9 1 9 5 3 0 9 2 1 8 6 1 1 7 3 8 1 9 3 2 6 1 1
7 9 3 1 0 5 1 1 8 5 4 8 0 7 4 4 6 2 3 7 9 9 6 2 7 4 9 5 6 7 3 5 1 8 8 5 7 5 2 7 2 4 8 9 1 2 2 7 9 3 8 1 8 3 0
1 1 9 4 9 1 2 9 8 3 3 6 7 3 3 6 2 4 4 0 6 5 6 6 4 3 0 8 6 0 2 1 3 9 4 9 4 6 3 9 5 2 2 4 7 3 7 1 9 0 7 0 2 1 7
9 8 6 0 9 4 3 7 0 2 7 7 0 5 3 9 2 1 7 1 7 6 2 9 3 1 7 6 7 5 2 3 8 4 6 7 4 8 1 8 4 6 7 6 6 9 4 0 5 1 3 2 0 0 0
5 6 8 1 2 7 1 4 5 2 6 3 5 6 0 8 2 7 7 8 5 7 7 1 3 4 2 7 5 7 7 8 9 6 0 9 1 7 3 6 3 7 1 7 8 7 2 1 4 6 8 4 4 0 9
0 1 2 2 4 9 5 3 4 3 0 1 4 6 5 4 9 5 8 5 3 7 1 0 5 0 7 9 2 2 7 9 6 8 9 2 5 8 9 2 3 5 4 2 0 1 9 9 5 6 1 1 2 1 2
9 0 2 1 9 6 0 8 6 4 0 3 4 4 1 8 1 5 9 8 1 3 6 2 9 7 7 4 7 7 1 3 0 9 9 6 0 5 1 8 7 0 7 2 1 1 3 4 9 9 9 9 9 9 8
3 7 2 9 7 8 0 4 9 9 5 1 0 5 9 7 3 1 7 3 2 8 1 6 0 9 6 3 1 8 5 9 5 0 2 4 4 5 9 4 5 5 3 4 6 9 0 8 3 0 2 6 4 2 5
2 2 3 0 8 2 5 3 3 4 4 6 8 5 0 3 5 2 6 1 9 3 1 1 8 8 1 7 1 0 1 0 0 0 3 1 3 7 8 3 8 7 5 2 8 8 6 5 8 7 5 3 3 2 0
8 3 8 1 4 2 0 6 1 7 1 7 7 6 6 9 1 4 7 3 0 3 5 9 8 2 5 3 4 9 0 4 2 8 7 5 5 4 6 8 7 3 1 1 5 9 5 6 2 8 6 3 8 8 2
3 5 3 7 8 7 5 9 3 7 5 1 9 5 7 7 8 1 8 5 7 7 8 0 5 3 2 1 7 1 2 2 6 8 0 6 6 1 3 0 0 1 9 2 7 8 7 6 6 1 1 1 9 5 9
0 9 2 1 6 4 2 0 1 9 8 9

Program listing.-

```
*LBL "PI1K"
*LBL A
" PI -?-"
AVIEW
CLRG
FIX 3
4
STO 09
E5
ST/ Y
STO 04
X^2
STO 05
X<>Y
427
+
STO 02
239
X^2
STO 07
LASTX
E2
*
STO 13
RDN
X^2
STO 08
94 E-5
STO 11
14.0139
STO 12
25
STO 10
*LBL 00
RCL 11
ST+ 12
RCL 12
RND
STO 00
RCL 07
RCL 02
INT
ENTER^
ST* Z
2
-
ST- Z
*
RCL 10
*
STO 06
CLX
STO 01
X<>Y
RCL 13
*
ENTER^
GTO 02
*LBL 01
RCL 06
ST/ Z
MOD
X<>Y
INT
X<>Y
RCL 04
ST* Z
*
ENTER^
*LBL 02
RCL 06
ST/ Z
MOD
STO 03
RDN
INT
+
RCL 05
ST- Y
X<>Y
RCL IND 00
+
X>0?
ISG 01
*LBL 03
X<0?
+
RCL 01
RCL 04
ST/ Z
*
ENTER^
*LBL 02
RCL 08
ST/ Z
MOD
R^
INT
LASTX
FRC
RDN
+
X<>Y
INT
RCL 04
ST* T
ST* Z
*
STO IND 00
RDN
ENTER^
*LBL 03
RCL 08
ST/ Z
MOD
X<>Y
INT
ST+ IND 00
RDN
+
STO 01
RCL 03
RCL 04
*
ENTER^
ISG 00
GTO 01
DSE 02
GTO 00
4096 E-7
STO 08
1439.00006
STO 02
837 E-6
STO 11
115.115
STO 12
80
STO 13
5 E6
STO 07
.75
STO 06
*LBL "Q"
```

145	RCL 11
146	ST+ 12
147	RCL 12
148	RND
149	STO 00
150	STO 03
151	SF 00
152	*LBL 05
153	RCL 02
154	INT
155	ENTER^
156	ENTER^
157	*LBL 02
158	2
159	-
160	ST* Z
161	RCL 10
162	ST* Z
163	X<>Y
164	*
165	2
166	ST- L
167	CLX
168	LASTX
169	ST* T
170	ST- Y
171	RDN
172	*
173	R^
174	ST+ T
175	X^2
176	R^
177	+
178	+
179	FC? 00
180	GTO 02
181	RCL 13
182	*
183	3
184	DSE 02
185	GTO 03
186	*LBL 02
187	RCL 07
188	*
189	RCL 06
190	*LBL 03
191	X<>Y
192	RDN
193	/
194	STO 01
195	CLX
196	R^
197	ENTER^
198	GTO 09
199	*LBL 08
200	RCL 01
201	ST/ Z
202	MOD
203	X<>Y
204	INT
205	X<>Y
206	RCL 04
207	ST* Z
208	*
209	ENTER^
210	*LBL 09
211	RCL 01
212	ST/ Z
213	MOD
214	RDN
215	INT
216	+
217	RCL IND 00
218	-
219	X>0?
220	GTO 02
221	DSE 00
222	*LBL 03
223	DSE IND 00
224	ISG 00
225	RCL 05
226	+
227	*LBL 02
228	STO IND 00
229	R^
230	RCL 04
231	*
232	ENTER^
233	ISG 00
234	GTO 08
235	RCL 03
236	STO 00
237	FS?C 00
238	GTO 05
239	CLX
240	ENTER^
241	DSE 02
242	FS? 00
243	GTO 04
244	*LBL 11
245	X<> IND 00
246	RCL 04
247	/
248	FRC
249	LASTX
250	INT
251	RCL 08
252	*
253	FRC
254	LASTX
255	INT
256	ST+ IND 00
257	RDN
258	X<>Y
259	RCL 05
260	ST* T
261	ST* Z
262	*
263	RCL 08
264	*
265	FRC
266	X<>Y
267	LASTX
268	INT
269	R^
270	+
271	RCL 05
272	-
273	+
274	X>0?
275	ISG IND 00
276	X>0?
277	GTO 03
278	RCL 05
279	+
280	*LBL 03
281	ISG 00
282	GTO 11
283	GTO “Q”
284	*LBL 04
285	RCL 03
286	STO 00
287	RCL 10
288	X^2
289	3
290	Y^X
291	LASTX
292	*
293	STO 08
294	CLX
295	*LBL 13
296	RCL IND 00
297	X<>Y
298	RCL 04
299	ST/ Z
300	*

```
ENTER^
*LBL 02
RCL 08
ST/ Z
MOD
R^
INT
LASTX
FRC
RDN
+
X<>Y
INT
RCL 04
ST* T
ST* Z
*
STO IND 00
RDN
ENTER^
*LBL 03
RCL 08
ST/ Z
MOD
X<>Y
INT
ST+ IND 00
RDN
+
ISG 00
GTO 13
114.013
STO 00
215
STO 03
CLX
*LBL 06
RCL IND 03
+
RCL IND 00
-
0
X<>Y
X<0?
X>0?
GTO 02
RCL 05
+
DSE Y
*LBL 02
STO IND 00
RDN
DSE 03
DSE 00
GTO 06
BEEP
RTN
*LBL E
SF 21
CLA
FIX 0
14.114
STO 00
SF 29
RCL IND 00
ACX
ADV
CF 29
ISG 00
*LBL 07
XEQ 10
ISG 00
FS? 00
RTN
" "
XEQ 10
ADV
CLA
ISG 00
GTO 07
AVIEW
RTN
*LBL 10
RCL IND 00
RCL 04
/
INT
LASTX
FRC
RCL 04
XEQ 12
" "
XEQ 12
RTN
*LBL 12
*
RCL Y
X=0?
GTO 03
LOG
INT
*LBL 03
RCL 09
X<>Y
X=Y?
GTO 02
-
0
*LBL 14
ARCL X
DSE Y
GTO 14
*LBL 02
ARCL T
ACA
CLA
END
```

Extended precision: E to 2,900 places. *(by Ron Knapp, PPCCJ V9N1 p12)*

This program is an abbreviated version designed to compute the decimal places of "e" to the greatest possible limit allowed in an HP-41CV or an HP-41C with a Quad Memory module. The program does the initialization including setting the SIZE to 294 data registers.

R01 shows the count-down number at all times. Originally this indicates the number of terms of the series necessary to obtain the accuracy desired. The number of terms yet to be computed is continuously displayed to allow the operator to know the progress of the computation. When the count-down number reaches zero the execution can proceed to the readout (or printout) routine, which displays 10 digits at a time, broken into two groups of five digits each, for easy reading. All leading and ending zeros are shown.

Instructions:

XEQ "E2900" Will take around 25 minutes at TURBO50 speed !
XEQ "R" To see/Print the results

```
01  LBL "R"        Readout results
02  FIX 0
03  CF 29
04  "2,"
05  AVIEW
06  4
07  ST+  03
08  LBL 06
09  CLA
10  SF 01
11  RCL IND 03
12   E5
13  /
14  FRC
15  LASTX
16  INT
17  LBL 07
18  ENTER^
19  ENTER^
20  4
21  X<>T
22  X=0?
23  GTO 08
24  LOG
25  INT
26  –
27  0
28  X=Y?
29  GTO 09
30  LBL 08
31  ARCL X
32  DSE  Y
33  GTO 08
34  LBL 09
35  ARCL  Z
36  FC?C 01
37  GTO 10
38  "/-  "              ; two spaces
39  R^
40   E5
41  *
42  GTO 07
43  LBL 10
44  AVIEW
45  ISG 03
46  GTO 06
47  END
```

Program listing. -

```
1	*LBL "E2900"
2	294
3	PSIZE
4	CF 01
5	CF 02
6	4.004
7	STO 00
8	1112
9	STO 01
10	E
11	STO 03
12	.293
13	STO 03
14	*LBL e
15	RCL 01
16	ENTER^
17	VIEW X
18	DSE 01
19	E10
20	X<>Y
21	ISG Z
22	*LBL 00
23	RCL 01
24	X<>Y
25	*
26	X>Y?
27	GTO 01
28	DSE 01
29	GTO 00
30	SF 01
31	ENTER^
32	*LBL 01
33	R^
34	LASTX
35	X<>Y
36	RCL 01
37	3
38	FC? 01
39	DSE X
40	*LBL 02
41	+
42	-
43	E
44	ENTER^
45	*LBL 03
46	X<> L
47	ST* Y
48	X<> L
49	ST+ Y
50	ST+ L
51	DSE Z
52	GTO 03
53	*
54	+
55	*LBL 04
56	E5
57	*
58	ENTER^
59	R^
60	ST/ Z
61	MOD
62	X<>Y
63	INT
64	E5
65	X>Y?
66	GTO 05
67	/
68	INT
69	E
70	ST- 00
71	X<>Y
72	ST+ IND 00
73	RDN
74	ST+ 00
75	CLX
76	LASTX
77	FRC
78	E5
79	*
80	LASTX
81	*LBL 05
82	*
83	X<> IND 00
84	LASTX
85	/
86	INT
87	ST+ Y
88	X<> L
89	FRC
90	X<>Y
91	E5
92	ST* Z
93	*
94	ENTER^
95	R^
96	ST/ Z
97	MOD
98	LASTX
99	RDN
100	X<>Y
101	INT
102	ST+ IND 00
103	CLX
104	+
105	+
106	ISG 00
107	GTO 04
108	X<>Y
109	/
110	RND
111	E
112	ST- 00
113	X<>Y
114	ST+ IND 00
115	R^
116	E-10
117	*
118	ST* 02
119	RCL 02
120	LASTX
121	X>Y?
122	SF 02
123	FS? 02
124	ST/ 02
125	E-3
126	RCL 00
127	FRC
128	FC?C 02
129	+
130	RCL 03
131	X<Y?
132	X<>Y
133	RDN
134	4
135	+
136	STO 00
137	FC?C 01
138	GTO e
139	END
```

EXTENDED PRECISION -- "e" to 2900 PLACES

2.71828 18284 59045 23536 02874 71352 66249 77572 47093 69995 95749 66967 62772 40766 30353 54759 45713 82178 52516 64274
27466 39193 20030 59921 81741 35966 29043 57290 03342 95260 59563 07381 32328 62794 34907 63233 82988 07531 95251 01901
15738 34187 93070 21540 89149 93488 41675 09244 76146 06680 82264 80016 84774 11853 74234 54424 37107 53907 77449 92069
55170 27618 38606 26133 13845 83000 75204 49338 26560 29760 67371 13200 70932 87091 27443 74704 72306 96977 20931 01416
92836 81902 55151 08657 46377 21112 52389 78442 50569 53696 77078 54499 69967 94686 44549 05987 93163 68892 30098 79312

77361 78215 42499 92295 76351 48220 82698 95193 66803 31825 28869 39849 64651 05820 93923 98294 88793 32036 25094 43117
30123 81970 68416 14039 70198 37679 32068 32823 76464 80429 53118 02328 78250 98194 55815 30175 67173 61332 06981 12509
96181 88159 30416 90351 59888 85193 45807 27386 67385 89422 87922 84998 92086 80582 57492 79610 48419 84443 63463 24496
84875 60233 62482 70419 78623 20900 21609 90235 30436 99418 49146 31409 34317 38143 64054 62531 52096 18369 08887 07016
76839 64243 78140 59271 45635 49061 30310 72085 10383 75051 01157 47704 17189 86106 87396 96552 12671 54688 95703 50354

02123 40784 98193 34321 06817 01210 05627 88023 51930 33224 74501 58539 04730 41995 77770 93503 66041 69973 29725 08868
76966 40355 57071 62268 44716 25607 98826 51787 13419 51246 65201 03059 21236 67719 43252 78675 39855 89448 96970 96409
75459 18569 56380 23637 01621 12047 74272 28364 89613 42251 64450 78182 44235 29486 36372 14174 02388 93441 24796 35743
70263 75529 44483 37998 01612 54922 78509 25778 25620 92622 64832 62779 33386 56648 16277 25164 01910 59004 91644 99828
93150 56604 72580 27786 31864 15519 56532 44258 69829 46959 30801 91529 87211 72556 34754 63964 47910 14590 40905 86298

49679 12874 06870 50489 58586 71747 98546 67757 57320 56812 88459 20541 33405 39220 00113 78630 09455 60688 16674 00169
84205 58040 33637 95376 45203 04024 32256 61352 78369 51177 88386 38744 39662 53224 98506 54995 88623 42818 99707 73327
61717 83928 03494 65014 34558 89707 19425 86398 77275 47109 62953 74152 11151 36835 06275 26023 26484 72870 39207 64310
05958 41166 12054 52970 30236 47254 92966 69381 15137 32275 36450 98889 03136 02057 24817 65851 18063 03644 28123 14965
50704 75102 54465 01172 72115 55194 86685 08003 68532 28183 15219 60037 35625 27944 95158 28418 82947 87610 85263 98139

55990 06737 64829 22443 75287 18462 45780 36192 98197 13991 47564 48826 26039 03381 44182 32625 15097 48279 87779 96437
30899 70388 86778 22713 83605 77297 88241 25611 90717 66394 65070 63304 52795 46618 55096 66618 56647 09711 34447 40160
70462 62156 80717 48187 78443 71436 98821 85596 70959 10259 68620 02353 71858 87485 69652 20005 03117 34392 07321 13908
03293 63447 97273 55955 27734 90717 83793 42163 70120 50054 51326 38354 40001 86323 99149 07054 79778 05669 78533 58048
96690 62951 19432 47309 95876 55236 81285 90413 83241 16072 26029 98330 53537 08761 38939 63917 79574 54016 13722 36187

89365 26053 81558 41587 18692 55386 06164 77983 40254 35128 43961 29460 35291 33259 42794 90433 72990 85731 58029 09586
31382 68329 14771 16396 33709 24003 16894 58636 06064 58459 25126 99465 57248 39186 56420 97526 85082 30754 42545 99376
91704 19777 80085 36273 09417 10163 43490 76964 23722 29435 23661 25572 50881 47792 23151 97477 80605 69672 53801 71807
76360 34624 59278 77846 58506 56050 78084 42115 29697 52189 08740 19660 90665 18035 16501 79250 46195 01366 58543 66327

Extended precision for Pi. (by Benoit Maag)

This section is a reproduction of the original article in the museum forum, see:

https://www.hpmuseum.org/forum/post-139434.html#pid139434

HP-41C Program / 41CL – DM41X

(X-functions only needed for memory sizing)

The program uses the formula: π = 2 + 1/3*(2 + 2/5*(2 + 3/7*(2 + …

n decimal precision obtained after INT(n/log(2)) iterations

Data stored as xxxxx.xxxxx – calculations done with 5 digits at a time. The fractional and integer part of the store number are separated and processed separately. The program is longer and slower as a result but memory use is maximized. Every iteration of i runs the multiplication by i from Rmax down to R03 and then the division by 2i+1 from R03 to Rmax.

Memory Usage

R00: indirect addressing register
R01: i, starting at INT(n/log(2)) and decreasing to 1
R02: number of last register of data
R03: x.xxxxx
R04 = Rmax: xxxxx.xxxxx (Rmax: last register of data)

Instructions

Nb of decimals desired (multiple of 10) XEQ "PI"

When the program ends (with a BEEP), the approximation of is stored in R03 ~ Rmax – nb of decimals = number of decimals desired + 5

Benckmarking:-

Notable absence is the V41 – TURBO test case, which of course will perform as good as the hosting PC machine is capable of performing.

Starting with the plain configuration:

HP-41C

# of Digits	# of iteration	# of registers	Time	Time (s)
15	49	2	5 min 46 s	346 s
25	83	3	14 min 15 s	855 s
45	149	5		
105	348	11	3 hrs 28 min 49 s	12,529 s

HP-41CL – TURBO50 Mode

# of Digits	# of iteration	# of registers	Time	Time (s)
15	49	2	23 s	23 s
25	83	3	54 s	54 s
45	149	5	2 min 32 s	152 s
105	348	11	12 min 21 s	741 s
255	847	26	1 hr 09 min 25 s	4,165 s

SWISSMICROS DM41X – Battery Power (*)

# of Digits	# of iteration	# of registers	Time	Time (s)
15	49	2	28 s	28 s
25	83	3	1 min 6 s	66 s
45	149	5	3 min 6 s	186 s
105	348	11	15 min 9 s	909 s
255	847	26	1hr 25 min 16 s	5,116 s

(*) printer module unplugged

SWISSMICROS DM41X – USB Power (*)

# of Digits	# of iteration	# of registers	Time	Time (s)
15	49	2	12 s	12 s
25	83	3	26 s	26 s
45	149	5	1 min 9 s	69 s
105	348	11	5 min 23 s	323 s
255	847	26	29 min 27 s	1,767 s

(*) printer module unplugged

Note: the printer module on the DM41X slows down the calculation significantly. For example, the calculation of 15 digits takes 74 seconds with the printer module plugged in, and just 28 seconds without it

Program Listing

```
LBL "Σ2PI"
ENTER
CLRG
5
+
2
STO 03
LOG
/
INT
STO 01
RDN
 E1
/
3
+
STO 00
STO 02
 E
+
PSIZE
0
LBL 00
RCL IND 00
FRC
XEQ 01
X<> IND 00
INT
 E5
/
XEQ 01
 E5
*
ST+ IND 00
RDN
DSE 00
RCL 00
3
X>Y?
GTO 02
RDN
RDN
GTO 00
LBL 01
RCL 01
*
X<>Y
 E5
/
+
INT
LAST X
FRC
RTN
LBL 02
0
ISG 00
FIX 5
LBL 05
 E5
*
RCL IND 00
INT
+
XEQ 03
X<> IND 00
FRC
+
 E5
*
XEQ 03
 E5
/
ST+ IND 00
RDN
ISG 00
FIX 5
RCL 00
RCL 02
X<Y?
GTO 04
RDN
RDN
GTO 05
LBL 03
ENTER
ENTER
RCL 01
ST+ X
ISG X
FIX 5
STO T
MOD
X<>Y
R^
/
INT
RTN
LBL 04
0 2
ST+ 03
DSE 01
GTO 06
BEEP
RTN
LBL 06
VIEW 01
DSE 00
0
GTO 00
END
```

Pi Decimals for the HP-41 (by Jean-Marc Baillard)
http://hp41programs.yolasite.com/pi.php

Overview

You place a positive integer n < 319 in the X-register, and your HP-41 returns 5.n decimals of PI , that is 5-digits per register up to 319 registers max or 1,595 digits.

Formula:

$$\pi = 2 + (1/3)\,(\,2 + (1/5)\,(\,2 + (3/7)\,(\,2 + \ldots\ldots\ \ (\,2 + k/(k+1)\,)\ \ldots.\,)\,)\,)$$

Program Listing

125 bytes / SIZE nnn+1

Data Registers: R00 = n ;
{R01 ... Rnn} = the decimals of PI in groups of 5 digits.

Flags: /

Subroutines: /

01 LBL "PIDIG"	17 E5	33 +	49 MOD	65 STO IND Z
02 CLRG	18 STO O	34 STO P	50 ST- Y	66 RDN
03 STO 00	19 ISG N	35 MOD	51 X<>Y	67 ISG Y
04 5	20 LBL 01	36 ST- 01	52 LASTX	68 GTO 02
05 *	21 RCL M	37 LASTX	53 /	69 DSE N
06 2	22 RCL O	38 ST/ 01	54 RCL O	70 GTO 01
07 LOG	23 ST+ X	39 CLX	55 ST* Z	71 E5
08 /	24 RCL 01	40 RCL O	56 X>Y?	72 ST+ 01
09 INT	25 +	41 *	57 GTO 03	73 ST+ 01
10 STO N	26 RCL N	42 LBL 02	58 ST- Y	74 ST/ 01
11 2	27 *	43 RCL IND Y	59 SIGN	75 RCL 00
12 RCL 00	28 STO 01	44 RCL N	60 ST- T	76 0.1
13 E3	29 LASTX	45 *	61 ST+ IND T	77 %
14 /	30 ST+ X	46 +	62 ST+ T	78 ISG X
15 +	31 ENTER	47 RCL X	63 LBL 03	79 CLA
16 STO M	32 SIGN	48 RCL P	64 RDN	80 END

STACK	INPUT	OUTPUT
X	n < 319	1.nnn

Example1: Calculate 5 x 8 = 40 decimals of PI

8 , XEQ "PIDIG" =>>> 1.008 ---Execution time = 11m14s---

-And we find in registers R01 thru R08: (add zeros on the left if need be)

3.14159 26535 89793 23846 26433 83279 50288 41971

All these decimals are exact !

Example2: Calculate 5 x 318 = 1590 decimals of PI

SIZE 319
318 XEQ "PIDIG" =>>>> 1.318 ---Execution time = 27m20s---
With V41 in Turbo Mode

And we get in registers R01 thru R318: (add zeros on the left if need be)

3.14159 26535 89793 23846 26433 83279 50288 41971 69399 37510 58209 74944 59230 78164 06286 20899 86280 34825 34211
70679 82148 08651 32823 06647 09384 46095 50582 23172 53594 08128 48111 74502 84102 70193 85211 05559 64462 29489
54930 38196 44288 10975 66593 34461 28475 64823 37867 83165 27120 19091 45648 56692 34603 48610 45432 66482 13393
60726 02491 41273 72458 70066 06315 58817 48815 20920 96282 92540 91715 36436 78925 90360 01133 05305 48820 46652
13841 46951 94151 16094 33057 27036 57595 91953 09218 61173 81932 61179 31051 18548 07446 23799 62749 56735 18857
52724 89122 79381 83011 94912 98336 73362 44065 66430 86021 39494 63952 24737 19070 21798 60943 70277 05392 17176
29317 67523 84674 81846 76694 05132 00056 81271 45263 56082 77857 71342 75778 96091 73637 17872 14684 40901 22495
34301 46549 58537 10507 92279 68925 89235 42019 95611 21290 21960 86403 44181 59813 62977 47713 09960 51870 72113
49999 99837 29780 49951 05973 17328 16096 31859 50244 59455 34690 83026 42522 30825 33446 85035 26193 11881 71010
00313 78387 52886 58753 32083 81420 61717 76691 47303 59825 34904 28755 46873 11595 62863 88235 37875 93751 95778
18577 80532 17122 68066 13001 92787 66111 95909 21642 01989 38095 25720 10654 85863 27886 59361 53381 82796 82303
01952 03530 18529 68995 77362 25994 13891 24972 17752 83479 13151 55748 57242 45415 06959 50829 53311 68617 27855
88907 50983 81754 63746 49393 19255 06040 09277 01671 13900 98488 24012 85836 16035 63707 66010 47101 81942 95559
61989 46767 83744 94482 55379 77472 68471 04047 53464 62080 46684 25906 94912 93313 67702 89891 52104 75216 20569
66024 05803 81501 93511 25338 24300 35587 64024 74964 73263 91419 92726 04269 92279 67823 54781 63600 93417 21641
21992 45863 15030 28618 29745 55706 74983 85054 94588 58692 69956 90927 21079 75093 02955 32116 53449 87202 75596
02364 80665 49911 98818 34797 75356 63698 07426 54252 78625 51818 41757 46728 90977

The Decimals of PI/E for the HP-41 (by Eckard Gehrke)

This section is a direct translation from the relevant sections of the chapter in the book " *HP-41 Sammlung*", pages 65, 66 and following. *Vieweg Programmbibliothek #23.*

3.3 The calculator program

The HP-41CV programmable calculator is used for the calculation. The HP-41 works with the RPN system, which is based on a bracket-free representation of all operations.

The HP-41 cannot define variables. It has numbered memories. A call is made with RCL nm, with a STO nm the number a is stored in register nm. For the used R00 holds i, R01 and R02 are needed for the loop counter j. 0 is stored in R03 and DR in R04. The following 81 memories R05 - R85 form ["R1"]

In these registers the successive elements are summed up to the registers R86 - R166 (R2) take in (b n), the division with D is handled in the registers R167 - R247.

The addressing of these registers is done indirectly with R 01 and R 02. For the subroutines addition and subtraction, the registers of R1 are called with RCL IND 01, those of R3 with RCL IND 02. The calculation of R 01 and R 02 is done in the subroutine loop counter. Only R 01 is needed for division. The register R0 (J) takes the remainder Registers. Register M ([ ) and N (\ ) are intended for ["M"] and ["N"] respectively.

The HP-41 can only jump to marks ("labels"). These are indicated in the diagrams with circles. For the labels NFG, ADD, DIV and SUB the labels 02, 03, 06 and A are used. Subroutines are executed with XEQ. On an RTN, the computer returns to the line following the subroutine call. Simple jumps are made with GTO.

For questions answered "no", the computer skips a line. The loop control is done with ISG and DSE. For i the result is: R 00 has the initial value 1.081 (a,b). If the computer comes to an ISG instruction, a is increased by 1: 2.081. If a>b, the computer skips one line. With a DSE instruction, a is decreased by 1. If a<b, one line is line is skipped.

With the help of lines 02 - 05 the calculator shows during the calculation

"PI=?" in the display. The rest of the program can be with the diagrams and the remarks on the basis of the commented program printout. program printout. To save memory space during the calculation, the output program has been separated. First the last digit is rounded, with LBL 00 the output begins the output. A diagram is not given for this.

With SIZE 248 the memory registers are reserved. The display format must be set to FIX 0. With XEQTPI the program is started. In the following 33 .4 hours the HP-41 calculates 800 decimals of pi. For this purpose, 580 subsequent elements an/D and 180 subsequent elements bn/D are calculated. The calculator switches on.

The number Pi program is then switched off. After switching on, SIZE 087 is used to create memory space for the output program. After reading in it is started with XEQ "OUT". When the printer is switched on, the following result is obtained:

Pi accurate to 800 decimals

```
π =
3,141592653    0454326648    7190702179
5897932384    2133936072    8609437027
6264338327    6024914127    7053921717
9502884197    3724587006    6293176752
1693993751    6063155881    3846748184
0582097494    7488152092    6766940513
4592307816    0962829254    2000568127
4062862089    0917153643    1452635608
9862803482    6789259036    2778577134
5342117067    0011330530    2757789609
9821480865    5488204665    1736371787
1328230664    2138414695    2146844090
7093844609    1941511609    1224953430
5505822317    4330572703    1465495853
2535940812    6575959195    7105079227
8481117450    3092186117    9689258923
2841027019    3819326117    5420199561
3852110555    9310511854    1212902196
9644622948    8074462379    0864034418
9549303819    9627495673    1598136297
6442881097    5188575272    7477130996
5665933446    4891227938    0518707211
1284756482    1830119491    3499999983
3378678316    2983367336    7297804995
5271201909    2440656643    1059731732
1456485669    0860213949    8160963186
2346034861    4639522473
```

The Number e

Let e(n) = Σ 1/k!, with k=0 to n.

Then | e-e(n) |<ε is valid with ε = (n+2)/(n+1)/(n+1)! . For ε=10^(-3002) one obtains n = 1143.

If one modifies the indicated procedure, one can achieve with the following algorithm that only the division subroutine and a register block are required.

The register assignment was made as follows: R00 - R301 (R1) contain e. The index j is stored in M ([), the divisor DR = n in N (\). R0 (J) takes up the remainder RE. The registers P (^) and a serve as temporary storage.

After a SIZE 302 the program can be started with XEQ "ZAHLE". After 6d 8h 24min the calculation is finished. The program **OUT** serves as output program. It can be loaded into the computer only after the program **ZAHLE** has been deleted. The addresses must be adapted to the register assignments. It results for e: = 2,718281828 .

Program listing.

```
01*LBL "PIZHAL"
02 248
03 PSIZE
04 "PI=?"
05 RCL d
06 AVIEW
07 STO d
08 CLRG
09 SF 00
10  E
11 STO 03
12 8 E10
13 STO 86
14 1.081
15 STO 00
16*LBL 00
17 25
18 STO 04
19 XEQ 02
20 XEQ 03
21 25
22 STO 04
23 XEQ 02
24 XEQ A
25 RCL 00
26 85
27 +
28 RCL IND X
29 X#0?
30 GTO 00
31 ISG 00
32 GTO 00
33 CF 00
34  E
35 STO 03
36 9.56 E11
37 STO 86
38 1.081
39 STO 80
40*LBL 01
41 57121
42 STO 04
43 XEQ 02
44 XEQ A
45 57121
46 STO 04
47 XEQ 02
48 XEQ 03
49 RCL 00
50 85
51 +
52 RCL IND X
53 X#0?
54 GTO 01
55*LBL 02
56 SF 01
57 XEQ 06
58 CF 21
59 RCL 03
60 STO 04
61 XEQ 06
62 2
63 ST+ 03
64 RTN
65*LBL 03
66 CF 02
67 XEQ B
68*LBL 04
60 0
70 FS?C 02
71  E
72 ST+ IND 02
73 RCL IND 02
74 RCL IND 01
75 STO M
76 +
77 STO IND 01
78  E10
79 X>Y?
80 GTO 05
81 ST- IND 01
82 RCL IND 02
83  E1
84 /
85 FRC
86 RCL M
87  E1
88 /
89 FRC
90 +
91 FRC
92 ENTER^
93 RND
94  E1
95 *
96 X<>Y
97  E1
98 *
99  -
100 ST- IND 01
101 SF 02
102*LBL 05
103  E
104 ST- 01
105 0
106 STO IND 02
107 DSE 02
108 GTO 04
109 RTN
110*LBL 06
111 CLA
112 166.166
113 RCL 00
114 FS? 01
115 85.085
116 +
117 STO 01
118*LBL 07
119 RCL IND 01
120 RCL 04
121 /
122 INT
123 STO M
124 RCL O
125  E10
126 *
127 RCL 04
128 /
129 INT
130 STO N
131 +
132 X<> IND 01
133 RCL 04
134 MOD
135 STO Z
136 RCL 04
137 /
138 RCL M
139 +
140 LASTX
141 -
142  E
143 X=Y?
144 ST- IND 01
145 RCL T
146 X<> O
147  E10
148 *
149 RCL 04
150 MOD
151 ST+ O
152 RCL 04
153 /
154 RCL N
155 +
156 LASTX
157 -
158  E
159 X=Y?
160 ST- IND 01
161 RCL O
162 RCL 04
163 X>Y?
164 GTO 08
165 MOD
166 STO O
167  E
168 ST+ IND 01
160*LBL 08
170 FC? 01
171 GTO 10
172 FC? 00
173 GTO 09
174 RCL IND 01
175 X#0?
176 GTO 09
177 FS?C 02
178 RTN
179 SF 02
180*LBL 09
181 RCL 01
182 81
183 +
184 RCL IND 01
```

```
185 STO IND Y
186*LBL 10
187 ISG 01
188 GTO 07
189 RTN
190*LBL A
191 CF 02
192 XEQ B
193*LBL 11
194 RCL IND 01
195 FC? 02
196 0
197 FS?C 02
198  E
199 -
200 RCL IND 02
201 -
202 STO IND 01
203 0
204 X<=Y?
205 GTO 12
206  E10
207 ST+ IND 01
208 SF 02
209*LBL 12
210  E
211 ST- 01
212 0
213 STO IND 02
214 DSE 02
215 GTO 11
216 RTN
217*LBL B
218 RCL 00
219 INT
220  E3
221 /
222 85.003
223 +
224 STO 01
225 162.162
226 +
227 STO 02
228 END
```

```
01*LBL "EZHAL"
02 302
03 PSIZE
04 CLRG
05 1143
06 STO N
07  E
08 STO 00
09*LBL 00
10 .301
11 STO M
12 0
13 STO O
14*LBL 01
15 RCL IND M
16 RCL N
17 /
18 INT
19  E10
20 X<>Y
21 STO P
22 X<>Y
23 RCL M
24 *
25 RCL N
26 /
27 INT
28 STO a
29 +
30 X<> IND M
31 RCL N
32 MOD
33 STO Z
34 RCL N
35 /
36 RCL ^
37 +
38 LASTX
39 -
40  E
41 X=Y?
42 ST- IND M
43 RCL T
44 X<> O
45  E10
46 *
47 RCL N
48 MOD
49 ST+ O
50 RCL N
51 /
52 RCL a
53 +
54 LASTX
55 -
56  E
57 X=Y?
58 ST- IND M
59 RCL O
60 RCL N
61 X>Y?
62 GTO 02
63 MOD
64 STO O
65  E
66 ST+ IND M
67*LBL 02
68 ISG [
69 GTO 01
70  E
71 ST+ 00
72 ST- \
73 RCL \
74 X>0?
75 GTO 00
76 OFF
77 END
```

```
01*LBL "OUT"
02 RCL 85
03  E9
04 /
05 INT
06 4
07 X>Y?
08 GTO 00
09  E9
10 ST+ 84
11*LBL 00
12 CF 28
13 FIX 9
14 CLA
15 RCL 05
16  E9
17 /
18 ARCL X
19 AVIEW
20 FIX 0
21 CF 29
22 6.084
23 STO T
24*LBL 01
25 RCL T
26 STO T
27 CLA
28 "0000"
29 ARCL IND T
30 RCL M
31 0
32 STO M
33 "`^^^^"
34 STO O
35 "`^^^"
36 RCL O
37 CLA
38 STO M
39 "`^^^^^^^^"
40 X<> Z
41 STO M
42 AVIEW
43 ISG T
44 GTO 01
45 CLST
46 END
```

Appendix. A few MCODE Listings.

1. Liu Hui formula.

Header	ACE8	089	"I"	
Header	ACE9	015	"U"	
Header	ACEA	008	"H"	
Header	ACEB	015	"U"	
Header	ACEC	009	"I"	
Header	ACED	00C	"L"	
LIUHUI	ACEE	391	PORT DEP:	shows "RUNNING"and init vars
	ACEF	08C	XQ	1 in {A,B}, 0 in N
	ACF0	070	->A870	[INIT] - lifts stack, sets DEC
	ACF1	0E0	SLCT Q	
	ACF2	25C	PT= 9	do eight times
LOOP9	ACF3	0A0	SLCT P	
	ACF4	04E	C=0 ALL	
	ACF5	35C	PT= 12	C=2
	ACF6	090	LD@PT- 2	
	ACF7	025	?NC XQ	2+result(k)
	ACF8	060	->1809	[AD1_10]
	ACF9	305	?NC XQ	sqr(2+result(k))
	ACFA	060	->18C1	[SQR13]
	ACFB	0E0	SLCT Q	
	ACFC	3D4	PT=PT-1	
	ACFD	314	?PT= 1	
	ACFE	01B	JNC +03	[NOT7]
PT=7	ACFF	2BE	C=-C-1 MS	sign change
	AD00	11E	A=C MS	ditto for 13-digit form
NOT7	AD01	394	?PT= 0	
	AD02	38B	JNC -15d	[LOOP9]
	AD03	04E	C=0 ALL	
	AD04	35C	PT= 12	
	AD05	1D0	LD@PT- 7	C=768
	AD06	190	LD@PT- 6	
	AD07	210	LD@PT- 8	
	AD08	130	LDI S&X	
	AD09	002	CON:	
	AD0A	13D	?NC XQ	
	AD0B	060	->184F	[MP1_10]
	AD0C	0E8	WRIT 3(X)	
	AD0D	3C1	?NC GO	Normal Function Return
	AD0E	002	->00F0	[NFRPU]
INIT	A870	18C	?FSET 11	
	A871	3B5	?C XQ	Stack lift
	A872	051	->14ED	[R_SUB]
	A873	2A9	?NC XQ	Show "RUNNING" - leaves F8 as-i
	A874	13C	->4FAA	[RUNMSG]
	A875	1A0	A=B=C=0	zero trinity
	A876	070	N=C ALL	k=0
	A877	2A0	SETDEC	
	A878	001	?NC GO	iniital sum = 1
	A879	062	->1800	[ADDONE]

2. Ramanujan 10-digit formula.

Header	ACBB	0B0	"0"	
Header	ACBC	031	"1"	**Ramanujan Approximation**
Header	ACBD	001	"A"	correct to 10 decimal digits
Header	ACBE	00D	"M"	
Header	ACBF	001	"A"	
Header	ACC0	012	"R"	Ángel Martin
RAMA10	**ACC1**	**18C**	**?FSET 11**	
	ACC2	3B5	?C XQ	Stack lift
	ACC3	051	->14ED	[R_SUB]
	ACC4	2A0	SETDEC	
	ACC5	04E	C=0 ALL	
	ACC6	35C	PT=12	
	ACC7	0D0	LD@PT- 3	C= 3 E-4
	ACC8	130	LDI S&X	
	ACC9	096	CON:	
	ACCA	2B6	C=-C-1 XS	
	ACCB	10E	A=C ALL	
	ACCC	04E	C=0 ALL	
	ACCD	2BE	C=-C-1 MS	
	ACCE	35C	PT=12	
	ACCF	0D0	LD@PT- 3	
	ACD0	150	LD@PT- 5	C= -3533
	ACD1	0D0	LD@PT- 3	
	ACD2	0D0	LD@PT- 3	
	ACD3	130	LDI S&X	
	ACD4	003	CON:	
	ACD5	261	?NC XQ	
	ACD6	060	->1898	[DV2_10]
	ACD7	001	?NC XQ	
	ACD8	060	->1800	[ADDONE]
	ACD9	04E	C=0 ALL	
	ACDA	130	LDI S&X	C= 355
	ACDB	355	CON:	
	ACDC	07C	RCR 4	
	ACDD	13D	?NC XQ	
	ACDE	060	->184F	[MP1_10]
	ACDF	04E	C=0 ALL	
	ACE0	130	LDI S&X	C= 113
	ACE1	113	CON:	
	ACE2	07C	RCR 4	
	ACE3	269	?NC XQ	
	ACE4	060	->189A	[DV1_10]
	ACE5	0E8	WRIT 3(X)	
	ACE6	3C1	?NC GO	Normal Function Return
	ACE7	002	->00F0	[NFRPU]

3. Viete's Formula. (next page)

Header	A654	*081*	*"A"*	
Header	A655	*014*	*"T"*	***Viete's Formula***
Header	A656	*005*	*"E"*	
Header	A657	*009*	*"I"*	
Header	A658	*016*	*"V"*	*Ángel Martin*
VIETA	**A659**	*391*	*PORT DEP:*	*shows "RUNNING"and init vars*
	A65A	*08C*	*XQ*	*1 in {A,B}, 0 in N*
	A65B	*070*	*->A870*	*[INIT] - lifts stack, sets DEC*
LOOP1	A65C	*089*	*?NC XQ*	*initial value*
	A65D	*064*	*->1922*	*[STSCR]*
	A65E	0F0	C<>N ALL	save N in M
	A65F	158	M=C ALL	*(N is used by [DSPCRG])*
	A660	0AE	A<>C ALL	*Mant. Sign & Exponent*
	A661	0DA	C=B M	*Mantissa value*
	A662	*099*	*?NC XQ*	*Sends C to display - sets HEX*
	A663	*02C*	*->0B26*	*[DSPCRG]*
	A664	198	C=M ALL	*restore things*
	A665	2A0	SETDEC	
	A666	03C	RCR 3	*bring C<3:5> to C.X*
	A667	226	C=C+1 S&X	*k+1*
	A668	106	A=C S&X	save it in A.X
	A669	1BC	RCR 11	*put it in C<3:5>*
	A66A	0A6	A<>C S&X	*bring it back to C.X*
	A66B	070	N=C ALL	*update register*
	A66C	1A0	A=B=C=0	*zero trinity*
LOOP2	A66D	3CC	?KEY	
	A66E	360	?C RTN	*bail out upon key depressed*
	A66F	04E	C=0 ALL	
	A670	35C	PT= 12	*builds "2" in C*
	A671	090	LD@PT- 2	
	A672	*025*	*?NC XQ*	*2+result(k)*
	A673	*060*	*->1809*	*[AD1_10]*
	A674	*305*	*?NC XQ*	*sqrt(2+ result(k))*
	A675	*060*	*->18C1*	*[SQR13]*
	A676	0B0	C=N ALL	*bring counter to C.X*
	A677	266	C=C-1 S&X	*decrease count*
	A678	070	N=C ALL	*update counter*
	A679	2E6	?C#0 S&X	*all done?*
	A67A	*39F*	*JC -13d*	*no, do next*
	A67B	*3D9*	*?NC XQ*	*{A,B} = {A,B} /2*
	A67C	*13C*	*->4FF6*	*[DIVBY2]*
	A67D	0E8	WRIT 3(X)	*term value*
	A67E	*0D1*	*?NC XQ*	*{Q,+} -> {C,M}*
	A67F	*064*	*->1934*	*[RCSCR]*
	A680	*149*	*?NC XQ*	*{A,B} = {A,B}*{C,M}*
	A681	*060*	*->1852*	*[MP2_13]*
	A682	0F8	READ 3(X)	*term value*
	A683	35C	PT= 12	
	A684	262	C=C-1 @PT	
	A685	2EE	?C#0 ALL	*was it equal to 1?*
	A686	*2B7*	*JC -42d*	*no, -> [LOOP]*
	A687	*239*	*?NC XQ*	
	A688	*060*	*->188E*	*[ONE_BY_X13]*
	A689	*025*	*?NC XQ*	*2x*
	A68A	*060*	*->1809*	*[AD1_10]*
	A68B	0E8	WRIT 3(X)	
	A68C	*3C1*	*?NC GO*	*Normal Function Return*
	A68D	*002*	*->00F0*	*[NFRPU]*

4. From Pi to e.

Header	A82C	085	"E"	From π to E
Header	A82D	032	"2"	
Header	A82E	009	"I"	Ángel Martin
Header	A82F	010	"P"	
PI2E	A830	379	PORT DEP:	shows "RUNNING"and init vars
	A831	03C	XQ	1 in {A,B}, 0 in N
	A832	070	->A870	[INIT] - lifts stack, sets DEC
LOOP	A833	3CC	?KEY	
	A834	360	?C RTN	
	A835	089	?NC XQ	current sum
	A836	064	->1922	[STSCR]
	A837	0B0	C=N ALL	n-1
	A838	1E1	?NC XQ	{A,B} = C+1
	A839	100	->4078	[INCC10]
	A83A	070	N=C ALL	n
	A83B	3B1	?NC XQ	Calculates Factorial
	A83C	060	->18EC	[XFT100]
	A83D	0E8	WRIT 3(X)	n!
	A83E	00E	A=0 ALL	clears MS and S&X
	A83F	269	?NC XQ	Puts p /2 in {M,C}
	A840	064	->199A	[PI/2]
	A841	1EE	C=C+C ALL	pi in {M,C}
	A842	0EE	B<>C ALL	moves it over to B
	A843	3C4	ST=0	
	A844	121	?NC XQ	Ln(π)
	A845	06C	->1B48	[LN13]
	A846	0B0	C=N ALL	n
	A847	13D	?NC XQ	n.Ln(π)
	A848	060	->184F	[MP1_10]
	A849	035	?NC XQ	π^n
	A84A	068	->1A0D	[EXP13]
	A84B	0F8	READ 3(X)	n!
	A84C	269	?NC XQ	π^n / n!
	A84D	060	->189A	[DV1_10]
	A84E	0E8	WRIT 3(X)	
	A84F	351	?NC XQ	Check error tolerance
	A850	128	>4AD4	[TOLER4]
	A851	2FE	?C#0 MS	negative? (passes tolerance)
	A852	03F	JC +07	yes, exit loop
	A853	0A9	?NC XQ	
	A854	064	->192A	[EXSCR] - {A,B} <-> {Q,+}
	A855	0F8	READ 3(X)	
	A856	025	?NC XQ	2+result(k)
	A857	060	->1809	[AD1_10]
	A858	2DB	JNC -37d	do next
EXITL	A859	0A9	?NC XQ	
	A85A	064	->192A	[EXSCR] - {A,B} <-> {Q,+}

	A85B	121	?NC XQ	
	A85C	06C	->1B48	[LN13]
	A85D	089	?NC XQ	
	A85E	064	->1922	[STSCR]
	A85F	00E	A=0 ALL	clears MS and S&X
	A860	269	?NC XQ	Puts p /2 in {M,C}
	A861	064	->199A	[PI/2]
	A862	1EE	C=C+C ALL	pi in {M,C}
	A863	0EE	B<>C ALL	moves it over to B
	A864	239	?NC XQ	1/ π
	A865	060	->188E	[ON/X13
	A866	0D1	?NC XQ	
	A867	064	->1934	[RCSCR]
	A868	149	?NC XQ	
	A869	060	->1852	[MP2-13]
	A86A	035	?NC XQ	final result
	A86B	068	->1A0D	[EXP13]
	A86C	0E8	WRIT 3(X)	
	A86D	3C1	?NC GO	Normal Function Return
	A86E	002	->00F0	[NFRPU]
	A86F	000	NOP	
INIT	A870	18C	?FSET 11	
	A871	3B5	?C XQ	Stack lift
	A872	051	->14ED	[R_SUB]
	A873	2A9	?NC XQ	Show "RUNNING" - leaves F8 as-i
	A874	13C	->4FAA	[RUNMSG]
	A875	1A0	A=B=C=0	zero trinity
	A876	070	N=C ALL	k=0
	A877	2A0	SETDEC	
	A878	001	?NC GO	iniital sum = 1
	A879	062	->1800	[ADDONE]

TOLER4	TOLER4	4AD4	01E	A=0 MS	absolute value
		4AD5	2A0	SETDEC	
expects error value stored		4AD6	04E	C=0 ALL	
in {A,B} in 13-digit form		4AD7	2BE	C=-C-1 MS	
		4AD8	35C	PT=12	
TOLER4		4AD9	050	LD@PT- 1	C= -1 E-9
TOLER4		4ADA	266	C=C-1 S&X	
TOLER4		4ADB	39C	PT= 0	
TOLER4		4ADC	050	LD@PT- 1	
TOLER4		4ADD	025	?NC GO	
TOLER4		4ADE	062	->1809	[AD1_10]

5. Wallis Formula (next page)

Header	AEAD	*093*	*"S"*	
Header	AEAE	*009*	*"I"*	***Wallis Formula Pi***
Header	AEAF	*00C*	*"L"*	*n in X*
Header	AEB0	*00C*	*"L"*	
Header	AEB1	*001*	*"A"*	
Header	AEB2	*017*	*"W"*	*Ángel Martin*
WALLIS	AEB3	*2A9*	*?NC XQ*	*Show "RUNNING" - leaves F8 as-i*
	AEB4	*13C*	*->4FAA*	*[RUNMSG]*
	AEB5	1A0	A=B=C=0	*zero trinity*
	AEB6	*001*	*?NC XQ*	
	AEB7	*060*	*->1800*	*[ADDONE]*
	AEB8	128	WRIT 4(L)	*initial term = 1*
	AEB9	0F8	READ 3(X)	
	AEBA	05E	C=0 MS	no negatives
	AEBB	2EE	?C#0 ALL	zero input?
	AEBC	3A0	?NC RTN	yes, bail out
	AEBD	*361*	*?NC XQ*	*(this includes SETDEC)*
	AEBE	*050*	*->14D8*	*[CHK_NO_S]*
LOOP	AEBF	070	N=C ALL	*update counter*
	AEC0	3CC	?KEY	
	AEC1	360	?C RTN	*bail out upon key depressed*
	AEC2	10E	A=C ALL	
	AEC3	*135*	*?NC XQ*	*n^2*
	AEC4	*060*	*->184D*	*[MP2_10]*
	AEC5	04E	C=0 ALL	
	AEC6	35C	PT= 12	***C=4***
	AEC7	110	LD@PT- 4	
	AEC8	*13D*	*?NC XQ*	*4n^2*
	AEC9	*060*	*->184F*	*[MP1_10]*
	AECA	*089*	*?NC XQ*	*intermediate result*
	AECB	*064*	*->1922*	*[STSCR]*
	AECC	*009*	*?NC XQ*	*4n^2 - 1*
	AECD	*060*	*->1802*	*[SUBONE]*
	AECE	*0D1*	*?NC XQ*	*4n^2 to {C,M}*
	AECF	*064*	*->1934*	*[RCSCR]*
	AED0	*24D*	*?NC XQ*	*[M,C} / {A,B}*
	AED1	*060*	*->1893*	*[X/Y13]*
	AED2	138	READ 4(L)	*previous result*
	AED3	*13D*	*?NC XQ*	
	AED4	*060*	*->184F*	*[MP1_10]*
	AED5	128	WRIT 4(L)	*updated result*
	AED6	0B0	C=N ALL	*get current iteration*
	AED7	10E	A=C ALL	*holds sign and S&X*
	AED8	02E	B=0 ALL	*clears it*
	AED9	0FA	B<>C M	*holds 13-digit mant*
	AEDA	*009*	*?NC XQ*	*13-digit form*
	AEDB	*060*	*->1802*	*[SUBONE]*
	AEDC	2EE	?C#0 ALL	*all done?*
	AEDD	*317*	*JC -30d*	*[LOOP]*
EXIT	AEDE	138	READ 4(L)	
	AEDF	10E	A=C ALL	
	AEE0	*01D*	*?NC XQ*	*2x*
	AEE1	*060*	*->1807*	*[AD2_10]*
	AEE2	*331*	*?NC GO*	*Overflow, DropST, FillXL & Exit*
	AEE3	*002*	*->00CC*	*[NFRX]*

5 . From e to pi

Header	A87A	089	"I"	
Header	A87B	010	"P"	
Header	A87C	032	"2"	
Header	A87D	005	"E"	Ángel Martin
E2PI	A87E	379	PORT DEP:	shows "RUNNING"and init vars
	A87F	03C	XQ	1 in {A,B}, 0 in N
	A880	070	->A870	[INIT] - lifts stack, sets DEC
	A881	035	?NC XQ	e
	A882	068	->1A0D	[EXP13]
	A883	0E8	WRIT 3(X)	10-digit e
	A884	089	?NC XQ	e as 13-digit value
	A885	064	->1922	[STSCR]
	A886	001	?NC XQ	e+1
	A887	060	->1800	[ADDONE]
	A888	0A9	?NC XQ	e
	A889	064	->192A	[EXSCR] - {A,B} <-> {Q,+}
	A88A	009	?NC XQ	e-1 in {A,B}
	A88B	060	->1802	[SUBONE]
	A88C	0D1	?NC XQ	e+1 to {C,M}
	A88D	064	->1934	[RCSCR]
	A88E	275	?NC XQ	(e-1)/(e+1)
	A88F	060	->189D	[DV2-13]
	A890	070	N=C ALL	required by [ATAN1]
	A891	3C4	ST=0	skips [TRGSET]
	A892	048	SETF 4	**result in RAD**
	A893	205	?NC XQ	it uses [SCR] as well
	A894	040	->1081	[ATAN1]
	A895	2BE	C=-C-1 MS	sign change
	A896	070	N=C ALL	store it in N
	A897	0F8	READ 3(X)	e
	A898	0F0	C<>N ALL	required by [ATAN1]
	A899	0E8	WRIT 3(X)	
	A89A	0B0	C=N ALL	
	A89B	3C4	ST=0	skips [TRGSET]
	A89C	048	SETF 4	**result in RAD**
	A89D	205	?NC XQ	it uses [SCR] as well
	A89E	040	->1081	[ATAN1]
	A89F	**11E**	**A=C MS**	bug or what?
	A8A0	0F8	READ 3(X)	
	A8A1	025	?NC XQ	2+result(k)
	A8A2	060	->1809	[AD1_10]
	A8A3	04E	C=0 ALL	
	A8A4	35C	PT= 12	**C=4**
	A8A5	110	LD@PT- 4	
	A8A6	13D	?NC XQ	
	A8A7	060	->184F	[MP1_10]
	A8A8	0E8	WRIT 3(X)	
	A8A9	3C1	?NC GO	Normal Function Return
	A8AA	002	->00F0	[NFRPU]

6. Erdós-Borwein constant.

Header	A6E4	*082*	*"B"*	*Erdós-Borwein constant*
Header	A6E5	*005*	*"E"*	*Ángel Martin*
EB	**A6E6**	*391*	*PORT DEP:*	*shows "RUNNING"and init vars*
	A6E7	*08C*	*XQ*	*1 in {A,B}, 0 in N*
	A6E8	*070*	*->A870*	*[INIT] - lifts stack, sets DEC*
LOOP	A6E9	3CC	?KEY	*converges in 30 iterations*
	A6EA	360	?C RTN	
	A6EB	*089*	*?NC XQ*	*current sum*
	A6EC	*064*	*->1922*	*[STSCR]*
	A6ED	0B0	C=N ALL	*n-1*
	A6EE	*1E1*	*?NC XQ*	*{A,B} = C+1*
	A6EF	*100*	*->4078*	*[INCC10]*
	A6F0	070	N=C ALL	n
	A6F1	04E	C=0 ALL	
	A6F2	35C	PT=12	*builds "2" in C*
	A6F3	090	LD@PT- 2	
	A6F4	084	CLRF 5	*Natural Ln*
	A6F5	*115*	*?NC XQ*	
	A6F6	*06C*	*->1B45*	*[LN10]*
	A6F7	0B0	C=N ALL	
	A6F8	*13D*	*?NC XQ*	*n.Ln(2)*
	A6F9	*060*	*->184F*	*[MP1_10]*
	A6FA	048	SETF 4	*substract one: e^x-1*
	A6FB	*035*	*?NC XQ*	*13-digit precision here*
	A6FC	*068*	*->1A0D*	*[EXP13]*
	A6FD	*239*	*?NC XQ*	*1/(2^x-1)*
	A6FE	*060*	*->188E*	*[ON/X13*
	A6FF	0E8	WRIT 3(X)	*n-th. term*
	A700	*351*	*?NC XQ*	*Check error tolerance*
	A701	*128*	*>4AD4*	*[TOLER4]*
	A702	2FE	?C#0 MS	*negative? (passes tolerance)*
	A703	***03F***	***JC +07***	*yes, exit loop*
	A704	*0A9*	*?NC XQ*	
	A705	*064*	*->192A*	*[EXSCR] - {A,B} <-> {Q,+}*
	A706	0F8	READ 3(X)	
	A707	*025*	*?NC XQ*	*new result(k)*
	A708	*060*	*->1809*	*[AD1_10]*
	A709	*303*	*JNC -32d*	*do next*
EXIT	A70A	*0A9*	*?NC XQ*	
	A70B	*064*	*->192A*	*[EXSCR] - {A,B} <-> {Q,+}*
	A70C	0AE	A<>C ALL	*Mant sign and exponent*
	A70D	0DA	C=B M	*Mantissa value*
	A70E	0E8	WRIT 3(X)	
	A70F	*3C1*	*?NC GO*	*Normal Function Return*
	A710	*002*	*->00F0*	*[NFRPU]*