

The approximate perimeter of an ellipse, according to Ramanujan

(Implemented in User RPL for the hp50g)

(and the 11C, 15C, 16C, 20s, 28C,S 32s, 33s, 35s, 39g, 42s, 48, and 95LX)

(...things got out of hand, obviously...)

D. A. Burkett (0db)

4 December 2024

Contents

1	Introduction	1
2	50g program ELPRA	4
3	Ellipse perimeter with elliptic integral $E(k)$	5
4	Ellipse segment arc length via direct integration	6
5	Srinavasa Ayengar Ramanujan	10
A	About the programs	11
B	HP-11C Ramanujan approximation	12
C	HP-15C Ramanujan approximation	13
D	HP-16C Ramanujan approximation	14
E	HP-20S Ramanujan approximation	15
F	HP-28 Ramanujan approximation	17
G	HP 32s Ramanujan approximation	17
H	HP 33s, 35s Ramanujan approximation	18
I	HP 39g Ramanujan approximation	19
J	HP 42s Ramanujan approximation	20
K	HP 48 series Ramanujan approximation	22
L	HP 95LX Ramanujan approximation	22
M	Ellipse perimeter test cases	22
N	References	23

1: Introduction

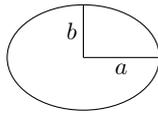
The Indian mathematician Srinivasa Ramanujan (1887–1920) developed a few formulas to approximate the perimeter of an ellipse. In this paper I implement the most accurate of those formulas as a User RPL program for the hp50g and summarize the program performance, including reduced accuracy for thin ellipses. Programs to calculate the Ramanujan approximation for several HP calculators are given in the appendices. The closed-form expression for the ellipse perimeter using the elliptic integral $E(k)$ is also implemented for the 50g, using a Chebyshev polynomial approximation for $E(k)$ from the literature.

An experimental 50g program is given to find the length of a segment of an ellipse, using the built-in numerical integration function f , and the performance is tested. This mostly illustrates the deficiencies of implementing special functions with integral definitions.

These programs are to be used with Numeric, Approx, and Radians modes.

As usual with these papers of mine, there is no original work save for the program code and some trivial algebraic manipulations. My sources are given in the *References* section.

I learned of the Ramanujan approximation from reference [1] which in turn refers to a paper by Almkvist and Berndt [2]. That 24-page paper starts with a description and analysis of Gauss's work on the arithmetic-geometric mean (AGM), describes using the AGM to calculate π , then gets into the ellipse perimeter approximations, with a brief detour to the *Ladies Diary* and back. 54 additional references are also given.



For an ellipse with semi-major axis a and semi-minor axis b , the approximate perimeter P given by Ramanujan is

$$P \approx \pi(a+b) \left(1 + \frac{3\lambda^2}{10 + \sqrt{4-3\lambda^2}} + \frac{3\lambda^{10}}{2^{17}} \right), \quad \lambda = \frac{a-b}{a+b}, \quad a > b$$

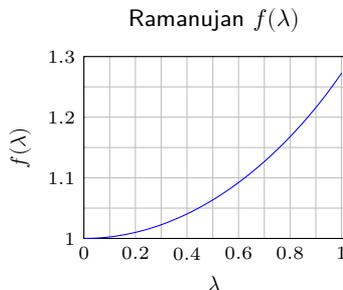
Note that if $a = b$ we have a circle, not an ellipse, and $\lambda = 0$ so $P = \pi(2a)$, which is just the circumference of the circle with diameter $2a$.

λ normalizes the eccentricity of the ellipse in the sense that (infinitely) many combinations of a and b result in the same value of λ . We can rewrite the equation above as

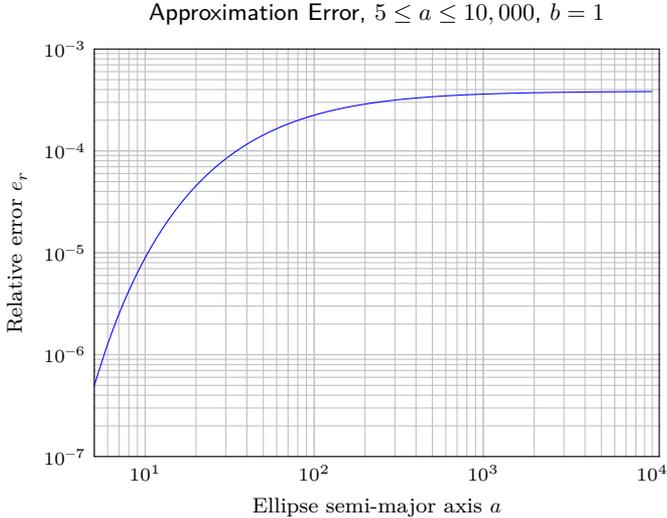
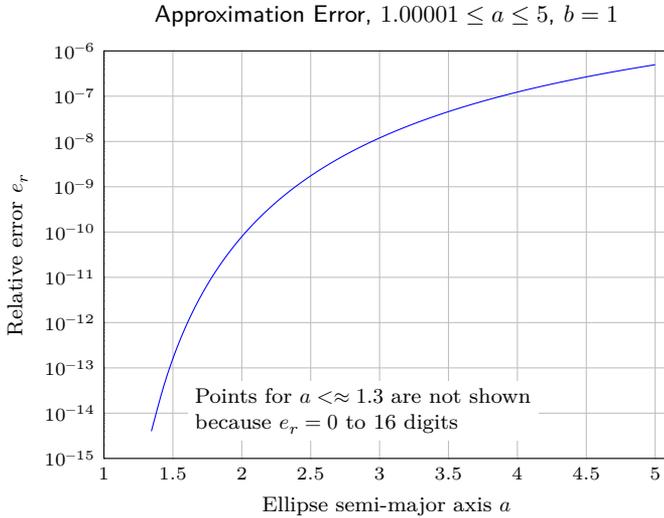
$$P \approx m f(\lambda), \quad m = \pi(a+b), \quad f(\lambda) = 1 + \frac{3\lambda^2}{10 + \sqrt{4-3\lambda^2}} + \frac{3\lambda^{10}}{2^{17}}$$

which makes it obvious that, for the Ramanujan approximation, the perimeter of any ellipse is the product of $f(\lambda)$, which is the same for many ellipses, and m , which specifies the parameters of a particular ellipse.

As mentioned above $\lambda = 0$ for $a = b$. In general, $0 \leq \lambda < 1$. λ only equals 1 when $b = 0$; in which case we don't have an ellipse. The plot below shows $f(\lambda)$. $f(1)$ is defined and is equal to 1835041/1441792 or about 1.272.



The plots below show the relative error of the Ramanujan's approximation over two different ranges of a with $b = 1$.



This is a fair approximation. The program is small and fast, and only uses basic arithmetic operations; no trig functions or logarithms and just a single square root. If we quantify the ellipse eccentricity as the ratio $r = a/b$, then we should get all 12 available digits for r up to about $r = 1.6$.

Even up to $r = 3$ we should have about 8 good digits. The relative error levels out to about $3 \cdot 10^{-3}$ for r up to 10^4 , which is a very skinny ellipse.

2: 50g program ELPRA

For calculation, the equation above is written as

$$P \approx k \left(1 + \frac{L}{10 + M} + c\lambda^{10} \right)$$

with

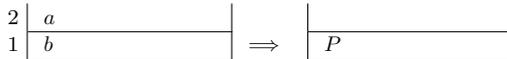
$$k = \pi(a + b), \quad \lambda = \frac{a - b}{a + b}, \quad L = 3\lambda^2, \quad M = \sqrt{4 - L}$$

and

$$c = 3/2^{17} \approx 2.28881835938 \cdot 10^{-5}$$

Nominally the approximation requires $a > b$. However, λ is only raised to even integer powers so implementations need not ensure $a > b$.

The program is named ELPRA (a severe contraction of “*ellipse perimeter, Ramanujan approximation*”) and has the stack I/O diagram



The perimeter of an ellipse with $a = 4$ and $b = 1$ is estimated with

$$4 \ 1 \ \text{ELPRA} \Rightarrow 17.1568414365$$

ELPRA	(123.5 bytes, checksum #4B0Eh, execution time \approx 46 mS)					
Code	1	2	3	4	5	Comment
«	b	a				
DUP2	b	a	b	a		Find $k = \pi(a + b)$
+ π *	k	b	a			
UNROT	b	a	k			Find λ ,
DUP2 -	$a - b$	b	a	k	...	$\lambda = (a - b)/(a + b)$
UNROT	b	a	$a - b$	k		
+ /	λ	k				
DUP SQ	λ^2	λ	k			Find $L = 3\lambda^2$
3. * DUP	L	L	λ	k		
4. SWAP - $\sqrt{\quad}$	M	L	λ	k	...	Find $M = \sqrt{4 - L}$
10. + /	Q	λ	k			$Q = L/(10 + M)$
1. +	$1 + Q$	λ	k			
SWAP 10. \wedge	λ^{10}	$1 + Q$	k			
2.28881835938E-5 *	$c\lambda^{10}$	$1 + Q$	k			
+	R	k				$R = 1 + Q + c\lambda^{10}$
*	P					P is the perimeter.
»						

The ellipses¹ ‘...’ in stack level 5 indicate that the level is temporarily used by the commands in the ‘Code’ column. Only 5 stack levels are used.

3: Ellipse perimeter with elliptic integral $E(k)$

It has long been known that the ellipse perimeter P with $a > b$ is given exactly by

$$P = 4aE(k), \quad k = \sqrt{1 - \left(\frac{b}{a}\right)^2}, \quad E(k) = \int_0^{\pi/2} \sqrt{1 - k^2(\sin \theta)^2} d\theta$$

where $E(k)$ is Legendre’s complete elliptic integral of the second kind.² Finding these perimeter approximations is an irresistible temptation for many people; even Kepler took a shot at it. In the end, these approximations all estimate $E(k)$ in one way or another. Usually the approximations are best for nearly-circular ellipses and accuracy degrades as the ellipse becomes more elongated ($a \gg b$), as does Ramanujan’s approximation.

There are many ways to calculate $E(k)$, but just for variety let’s try a Chebyshev approximation given by W. J. Cody in reference [8]. For real k the approximation is

$$E(k) \approx 1 + \sum_{i=1}^n c_i v^i + \ln\left(\frac{1}{v}\right) \sum_{i=1}^n d_i v^i, \quad v = 1 - k^2$$

Cody gives tables of the polynomial coefficients c_i and d_i for different values of n . His table Ia indicates that the maximum deviation from the true value of $E(k)$ for $n = 7$ is $2.18 \cdot 10^{-13}$, which is below the 50g minimum resolution of $1 \cdot 10^{-12}$. The coefficients are given with 15 significant digits, so I’ve rounded them to the 12 significant digits shown in the table below.

i	c_i	d_i
1	4.431 4719 3468 E-1	2.499 9999 8449 E-1
2	5.681 1568 1054 E-2	9.374 8806 2098 E-2
3	2.218 6220 6994 E-2	5.849 5029 7066 E-2
4	1.568 4770 0240 E-2	4.090 7482 1593 E-2
5	1.922 8438 9023 E-2	2.350 9160 2565 E-2
6	1.218 1948 1487 E-2	6.456 8224 7315 E-3
7	1.556 1874 4745 E-3	3.788 8648 7349 E-4

The approximation for $E(k)$ is implemented as program EECOM, below.

$$1 \left[k \right] \Rightarrow \left[E(k) \right]$$

¹No pun intended.

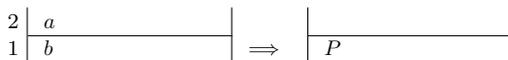
²And you could use John Keith’s *elliptic.zip* package at <https://www.hpcalc.org/details/9612>; it includes both a program to calculate ellipse perimeters using $E(k)$, as well as $E(k)$ itself and the other two complete elliptic integrals.

EECOM		(216 bytes, checksum C16Ah, $t \approx 0.15$ s)
Code	Comment	
«		
SQ 1 SWAP –	Find $v = 1 - k^2$	
[3.78886487349E-4 6.45682247315E-3	d_i coefficients	
2.35091602565E-2 4.09074821593E-2		
5.84950297066E-2 9.37488062098E-2		
.249999998449 0]		
OVER PEVAL	Find $P_d = \Sigma d_i v^i$	
OVER INV LN *	Find $\ln(1/v)P_d$	
[1.55618744745E-3 1.21819481487E-2	c_i coefficients	
1.92284389023E-2 1.56847700240E-2		
2.21862206994E-2 5.68115681054E-2		
.443147193468 0]		
ROT PEVAL	Find $P_c = \Sigma c_i v^i$	
+ 1 +	$E(k) = 1 + P_c + \ln(1/v)P_d$	
»		

I tested EECOM with 1000 values from $k = 0$ to $k = 0.999$. The maximum absolute error is about $\pm 8 \cdot 10^{-12}$, which exceeds the error of $2.18 \cdot 10^{-13}$ given by Cody. There are two reasons for this: the coefficients have been rounded down from 15 to 12 digits, and the 50g only has 12-digit precision.

EECOM does no input validation and requires $0 \leq k < 1$, although $E(1) = 1$; the program could be modified to handle this special case.

Program ELIPP listed below finds the ellipse perimeter P with EECOM. a and b may actually be entered in either order as ELIPP swaps the values as needed so $a > b$. For the test cases in Appendix M, ELIPP has a maximum relative error of about $9 \cdot 10^{-12}$.



ELIPP		(78 bytes, checksum 3836h, $t \approx 0.18$ s)
Code	Comment	
«		
DUP2 > «SWAP» IFT	Swap a and b so $a > b$, if needed	
DUP UNROT	Copy a	
/ SQ 1. SWAP – √ EECOM	Find k and $E(k)$	
* 4. *	Find $P = 4aE(k)$	
»		

4: Ellipse segment arc length via direct integration

A more interesting problem is to find the arc length P_s of a segment of an ellipse, which is a straight-forward application of basic calculus. If the ellipse is defined with polar coordinates (r, θ) then the arc length from $\theta = \alpha$

to $\theta = \beta$ with $\alpha < \beta$ is

$$P_s = \int_{\alpha}^{\beta} \sqrt{a^2 \sin^2 \theta + b^2 \cos^2 \theta} d\theta = b E \left(1 - \left(\frac{b}{a} \right)^2, \theta \right) \Big|_{\theta=\alpha}^{\theta=\beta}$$

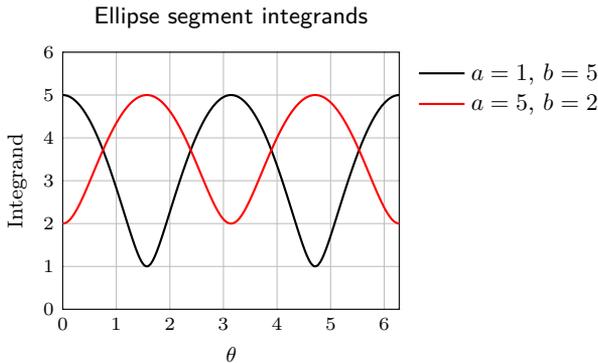
where a is 1/2 of the ellipse axis along the x-axis and b is 1/2 of the ellipse axis along the y-axis. Unlike the Ramanujan approximation above, we may have $b > a$.

$E(x, \theta)$ is the incomplete elliptic integral of the second kind; the common integral definition is

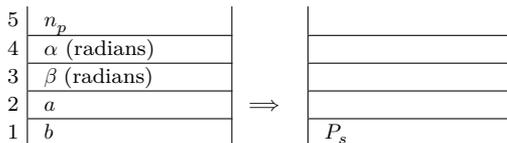
$$E(k, \theta) = \int_0^{\theta} \sqrt{1 - k^2 \sin^2 \varphi} d\varphi$$

Unfortunately the 50g does not implement the incomplete elliptic integrals, and I did not find any libraries at hpcalc.org. We can always try direct numerical integration, either with the expression for P_s or for $E(k, \theta)$, although that is almost never the most efficient method to evaluate a special function. We'll try the integral for P_s , which also makes a good example of using integration within a program, where the integrand includes parameters.

Before blindly numerically integrating a function, it is usually a good idea to have some notion of what the function looks like. From inspection the integrand of P_s has no singularities and is always non-negative. Since the arguments of both $\sin(\theta)$ and $\cos(\theta)$ are just θ , the integrand has a period of π radians. For our application we are interested in $0 \leq \theta \leq 2\pi$ so we will never have more than two cycles and our integrand is not terribly oscillatory. In other words, a pretty benign integrand. The plot below shows the integrands for $a = 1, b = 5$, and $a = 5, b = 2$.



The stack diagram for the program, EPSG1, is



where n_p is the number of digits of precision desired in P_s and can be set from 1 to 11. The code follows.

```

Program EPSG1                                (62.5 bytes, checksum #5F65h)
«
  Define the integrand function:
  « DUP COS ←b * SWAP SIN ←a * R→C ABS »
  Save the local variables.  $\gamma$  is the integrand function.
  → ←a ←b  $\gamma$ 
  «
  Save the display mode, then set the display mode to  $n_p$  digits:
  PUSH ROT SCI
  Evaluate the integral with limits  $\alpha$  and  $\beta$  on the stack:
  ' $\gamma(\tau)$ ' ' $\tau$ '  $\int$ 
  Restore the display mode:
  POP
  »»

```

The integrand is defined as a User RPL program and saved to local variable γ . It will be called repeatedly by \int to evaluate the integral, and \int puts a value of θ on the stack which γ can use as needed to evaluate the integrand. The parameters a and b must be saved as compiled local variables, so they will be visible to γ from the main program. γ finds $a \sin(\theta)$ and $b \cos(\theta)$ and converts these results to a complex number. That conversion is done so that ABS can be used to find calculate the value of the integrand. If z is a complex number $z = c + di$, then ABS finds $\sqrt{c^2 + d^2}$, which is just what we want when $c = a \sin(\theta)$ and $d = b \cos(\theta)$.

The PUSH command saves the system flags, to be restored by POP at the end of the program. The subsequent commands ROT SCI set the display mode to n_p digits, because \int uses the display mode setting as a precision target for the integration. Using n_p to specify the approximate accuracy of P_s allows us to trade off execution time for precision, without requiring the user to manually set the display mode before executing EPSG1.

All that remains is to actually evaluate the integral. The integrand function is put on the stack as ' $\gamma(\tau)$ ', as is the variable of integration ' τ '. The Greek letters γ and τ are used to try to avoid conflicts with existing global variables.

Note these additional characteristics of EPSG1:

- EPSG1 does no input validation. For example, if you swap α and β on the stack, the returned segment will be negative.
- All input arguments must be explicit real numeric constants. n_p will be rounded to an integer by SCI. EPSG1 is not listable.
- The system variable IERR is created (if necessary) and updated by \int , which is an estimate of the error bound for the integral. You can manually check IERR after running the program.
- Both α and β should be less than 2π radians, with $\alpha < \beta$. You can find the complete perimeter of the ellipse with $\alpha = 0$ and $\beta = 2\pi = 6.28318530718$; just use $2 \pi *$.

To use EPSG1 to estimate the perimeter segment from $\alpha = 0.1$ to $\beta = 2.9$, for the ellipse with $a = 5$ and $b = 2$ with $n_p = 11$:

11 0.1 2.9 5 2 EPSG1 \Rightarrow 10.7982469604

which is correct to 12 digits, according to Mathematica's numerical evaluation of the same integral. To find the total perimeter of the same ellipse, set $\alpha = 0$ and $\beta = 2\pi$:

11 0 2 π * 5 2 EPSG1 \Rightarrow 23.0131125956

which is in error by one digit in the last place; 6 should be 7. Execution time for this example is about 32.5 seconds.

The table below shows some test results for $a = 5$, $b = 2$, $\alpha = 0.1$, and $\beta = 3.1$, with different values of n_p . t_{eq} is the execution time in seconds, and the correct digits of P_s are underlined. For $n_p \geq 7$ all 12 digits are correct, so in this particular case increasing n_p above 7 only increases execution time. Note that the execution time sporadically jumps up by a factor of about 2 as n_p increases, while IERR decreases steadily and always overestimates the actual error.

EPSG1 accuracy and run time, n_p varies				
n_p	t_{eq}	P_s	Correct digits	IERR
1	0.58	<u>11.2718060662</u>	3	1.130
2	1.10	<u>11.2213344732</u>	5	$1.12 \cdot 10^{-1}$
3	1.10	<u>11.2213344732</u>	5	$1.12 \cdot 10^{-2}$
4	2.11	<u>11.2215095945</u>	6	$1.12 \cdot 10^{-3}$
5	4.13	<u>11.2215122014</u>	9	$1.12 \cdot 10^{-4}$
6	4.14	<u>11.2215122422</u>	10	$1.12 \cdot 10^{-5}$
7	8.17	<u>11.2215122494</u>	12	$1.12 \cdot 10^{-6}$
8	8.17	<u>11.2215122494</u>	12	$1.12 \cdot 10^{-7}$
9	8.18	<u>11.2215122494</u>	12	$1.12 \cdot 10^{-8}$
10	16.20	<u>11.2215122494</u>	12	$1.12 \cdot 10^{-9}$
11	16.21	<u>11.2215122494</u>	12	$1.12 \cdot 10^{-10}$

Another way to quantify execution time and accuracy is to fix n_p and b and vary a . The table below shows the results for $n_p = 7$, $b = 1$, $\alpha = 0$, and $\beta = 6.2$. For these values of α and β we are estimating almost the complete perimeter of the ellipse.

EPSG1 accuracy and run time, a varies			
a	t_{eq}	P_s	Correct digits
1.1	8.1	<u>6.51787967216</u>	10
1.5	8.1	<u>7.84941488507</u>	10
2	8.1	<u>9.60497637555</u>	8
3	16.1	<u>13.2809477045</u>	11
4	16.1	<u>17.0722427179</u>	10
5	16.1	<u>20.9246139821</u>	9
6	32.3	<u>24.8136532872</u>	11
7	32.3	<u>28.7266003348</u>	11
8	32.3	<u>32.6560787283</u>	10
9	32.3	<u>36.5975036768</u>	9
10	32.3	<u>40.5478643725</u>	9
20	65.9	<u>80.2758116967</u>	8
50	131.9	<u>199.992833092</u>	10

Finally, we can fix $n_p = 7$, $\alpha = 0$, $a = 5$, and $b = 1$, then vary β to quantify run time and accuracy as β increases, to include more of the integrand cycles.

EPSG1 accuracy and run time, a varies			
β	t_{eq}	P_s	Correct digits
0.5	4.1	<u>0.816551239826</u>	10
1.0	8.0	<u>2.54458646547</u>	10
1.5	8.1	<u>4.89881329292</u>	9
2.0	8.1	<u>7.33592964853</u>	10
2.5	8.1	<u>9.28803372160</u>	10
3.0	8.1	<u>10.3528187107</u>	10
3.5	8.1	<u>11.0000665390</u>	10
4.0	16.1	<u>12.4765675502</u>	11
4.5	16.2	<u>14.7032336766</u>	10
5.0	8.1	<u>17.1766435486</u>	8
5.5	16.1	<u>19.3183734250</u>	9
6.0	16.1	<u>20.6534613205</u>	8
6.28	32.2	<u>21.0068591032</u>	12

Overall, this performance is typical when a special function is calculated with an integral definition, which is essentially what we're doing here. Accurate results are possible (this is not always the case in general) but the execution time can be very long. A good program for the ellipse segment arc will have to wait for a good implementation of the incomplete elliptic integral of the second kind $E(k, \theta)$.

5: Srinavasa Aiyengar Ramanujan

Ramanujan was an Indian mathematician who did much of his mathematical work in Great Britain with Godfrey Hardy at Cambridge. Ramanujan is one of a few mathematicians who can be said to have some general popularity, which I think is due to the somewhat unusual circumstances of his personal and mathematical life. In his early years in school in India he did quite well until he came across a famous math textbook in high school, a synopsis of results in pure and applied mathematics. He worked his way completely through the book, checking the theorems and examples. Although he had a scholarship to a state university in India, he failed his first year by mostly ignoring all his subjects except mathematics. This failure resulted in his unemployment until he got a job as a clerk with the Madras Port Authority. During this period he continued his mathematical researches and began publishing some papers in local journals.

While clerking at the Port Authority he sent a letter to Hardy with some of his results. Hardy was impressed, and as an influential British mathematician Hardy was eventually able to get Ramanujan a scholarship and to convince him to come to Cambridge in 1914 (leaving his wife behind in India), both to work with him, and to fill in the substantial gaps in Ramanujan's mathematical knowledge — the same gaps in the synopsis earlier studied by Ramanujan. The notion of a rigorous proof was also not in Ramanujan's repertoire, possibly because, again, proofs were not a large part of the synopsis from which he had learned.

The culture shock for Ramanujan in Britain, as a high-caste Brahmin, was severe. There were other Indian students at Cambridge at the time, so he did make some friends who helped him learn and adapt to the curious ways of the British, but between the wet, frigid winters and the meat-heavy British diet (as a Brahmin, Ramanujan was a very strict vegetarian), he had a rough time. In spite of those difficulties Ramanujan published 25 papers in five years, some with Hardy, in number theory and the theory of functions. That's averaging a paper every five months and considering the quality of most of the papers, an impressive burst of activity.

Unfortunately Ramanujan contracted tuberculosis while in Britain, and his health was never robust to begin with. He returned to India in 1919 with the hopes that his health would improve in the more familiar climate, but nonetheless passed away in in April, 1920, at the age of 32. His funeral was sparsely attended, because he had broken one of the taboos of his particular Brahmin sect by traveling by sea. On his return to India he was too ill to travel to perform purification rites which would have removed the stigma.

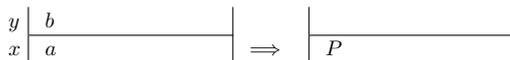
A: About the programs

The programs for the different calculators in the following sections all implement Ramnujan's approximation of the complete perimeter of the ellipse. None of the programs include validation of the input arguments and no error trapping is done. In particular, setting either or both of a and b to zero will raise a 'divide by 0' error.

Except for the 42s program, stack and register contents are overwritten. All of the programs were coded and tested on physical calculators.

Each program has been tested with a small set of test cases to catch gross coding errors. In other words, you're really using these programs at your own risk. For the calculators with very limited memory you might be better off with a simpler, less accurate approximation, particularly if you accuracy needs are modest and your ellipses are moderately circular. Appendix M gives a list of test cases which may be used to test accuracy with a particular calculator.

I am sure that the programs could be further optimized for size and speed, although I have made some effort to write reasonably efficient code. For all of the RPN calculators, the stack I/O diagram is



where a is the semi-major axis, b is the semi-minor axis, and P is the perimeter. For the non-RPN calculators, the I/O is documented. Actually, a and b inputs may be swapped with the Ramanujan approximation. Geometrically, this is equivalent to rotating the ellipse 90°; the perimeter remains the same.

I tend not to use the `LASTx` or `LAST` commands if those commands can be disabled. This allows the programs to work correctly regardless of the `LASTx` status. Some of the programs would be slightly smaller using those commands.

Stack diagrams are shown for some of the programs. A ‘?’ in a stack level or storage register means the contents when the program started executing.

B: HP-11C Ramanujan approximation

All four stack levels and register R0 are used.

Line	Instruction	Keycode	x	y	z	t
001	LBL A	42,21,11	b	a	?	?
002	STO 0	44 0	b	a	?	?
003	X<>Y	34	a	b	?	?
004	STO+ 0	44,40,0	a	b	?	?
005	—	30	$b - a$?	?	?
006	RCL 0	45 0	$a + b$	$b - a$?	?
007	/	10	$-\lambda$?	?	?
008	3	3	3	$-\lambda$?	?
009	X<>Y	34	$-\lambda$	3	?	?
010	x^2	43 11	λ^2	3	?	?
011	\times	20	L	?	?	?
012	LST x	43 36	λ^2	L	?	?
013	8	8	8	λ^2	L	?
014	/	10	$\lambda^2/8$	L	?	?
015	5	5	5	$\lambda^2/8$	L	?
016	y^x	14	A	L	?	?
017	3	3	3	A	L	?
018	\times	20	$3A$	L	?	?
019	4	4	4	$3A$	L	?
020	/	10	B	L	?	?
021	X<>Y	34	L	B	?	?
022	ENTER	36	L	L	B	?
023	ENTER	36	L	L	L	B
024	4	4	4	L	L	B
025	X<>Y	34	L	4	L	B
026	—	30	$4 - L$	L	B	B
027	\sqrt{x}	11	M	L	B	B
028	1	1	1	M	L	B
029	10^x	13	10	M	L	B
030	+	40	$M + 10$	L	B	B
031	/	10	D	B	B	B
032	+	40	$D + B$	B	B	B
033	1	1	1	$D + B$	B	B
034	+	40	E	B	B	B
035	RCL 0	45 0	$a + b$	E	B	B
036	\times	20	$E(a + b)$	B	B	B
037	π	42 16	π	$E(a + b)$	B	B
038	\times	20	P	B	B	B
039	RTN	43 32	P	B	B	B

These abbreviations are used in the stack diagram.

$$\begin{array}{lll}
 A = \lambda^{10}/2^{15} & B = 3\lambda^{10}/2^{17} & L = 3\lambda^2 \\
 M = \sqrt{4-L} & D = L/(10+M) & E = D + B + 1
 \end{array}$$

- Lines 002–007 find $-\lambda$ and save $a + b$ in R0 for use later. $-\lambda$ need not be converted to λ since it is only raised to even powers later.
- Lines 008–011 find $L = 3\lambda^2$.
- Lines 011–020 find $B = 3\lambda^{10}/2^{17}$ in a convoluted way to reduce program lines. B is actually calculated as $(3/4)(\lambda^2/8)^5$ since λ^2 is already available, and all the numeric constants become single digits. Numerically, this can be sketchy, but some testing shows that we get away with it this time.
- Lines 021–027 find $M = \sqrt{4-L}$. The double ENTER at lines 022 and 023 is necessary to get two copies of L on the stack, followed by the numeric constant 4.
- Lines 028 and 029 enter 10 on the stack without pushing B off the stack. Otherwise, explicitly entering 1 0 (for 10) would take two stack levels. Those two lines could also be replaced with EEX 1.

Execution time is about 4.5 seconds. To estimate the accuracy of the calculation I tested 72 cases with $b = 1$ and a ranging from 1 to 10,000. The relative error for all 72 cases was zero.

C: HP-15C Ramanujan approximation

The program listing below gives the code for the 15C version of the Ramanujan approximation. The following abbreviations are used in the stack diagram; these are *not* named 15C storage registers.

$$\begin{array}{lll}
 \lambda = \frac{a-b}{a+b} & L = 3\lambda^2 & B = \frac{3\lambda^2}{2^{17}} \\
 M = \sqrt{4-L} & D = \frac{L}{10+M} & E = D + B + 1
 \end{array}$$

Lines 12–20 find B in a complicated way to avoid entering $3/2^{17}$ as a numeric constant, which reduces the program size. The calculation is arranged so that all numeric constants are single digits. Lines 28–29 enter 10 on the stack, without pushing B off. Some direct register arithmetic is used to reduce code size slightly.

The program uses all four stack registers and register R0.

Line	Code	Keycode	x	y	z	t	R0
001	LBL A	42,21,11	b	a	?	?	?
002	STO 0	44 0	b	a	?	?	b
003	X<>Y	34	a	b	?	?	b
004	STO-0	44,30,0	a	b	?	?	$b-a$

Line	Code	Keycode	x	y	z	t	R0
005	+	40	$a + b$?	?	?	$b - a$
006	STO \div 0	44,10, 0	$a + b$?	?	?	$-\lambda$
007	X<> 0	42, 4, 0	$-\lambda$?	?	?	$a + b$
008	3	3	3	$-\lambda$?	?	$a + b$
009	X<>Y	34	$-\lambda$	3	?	?	$a + b$
010	x^2	43 11	λ^2	3	?	?	$a + b$
011	\times	20	L	?	?	?	$a + b$
012	LST x	43 36	λ^2	L	?	?	$a + b$
013	8	8	8	λ^2	L	?	$a + b$
014	\div	10	$\lambda^2/8$	L	?	?	$a + b$
015	5	5	5	$\lambda^2/8$	L	?	$a + b$
016	y^x	14	$\lambda^{10}/2^{15}$	L	?	?	$a + b$
017	3	3	3	$\lambda^{10}/2^{15}$	L	?	$a + b$
018	\times	20	$3\lambda^{10}/2^{15}$	L	?	?	$a + b$
019	4	4	4	$3\lambda^{10}/2^{15}$	L	?	$a + b$
020	/	10	B	L	?	?	$a + b$
021	X<>Y	21	L	B	?	?	$a + b$
022	ENTER	36	L	L	B	?	$a + b$
023	ENTER	36	L	L	L	B	$a + b$
024	4	4	4	L	L	B	$a + b$
025	X<>Y	34	L	4	L	B	$a + b$
026	-	30	$4 - L$	L	B	B	$a + b$
027	\sqrt{x}	11	M	L	B	B	$a + b$
028	EEX	26	1E0	M	L	B	$a + b$
029	1	1	10	M	L	B	$a + b$
030	+	40	$M + 10$	L	B	B	$a + b$
031	/	10	D	B	B	B	$a + b$
032	+	40	$D + B$	B	B	B	$a + b$
033	1	1	1	$D + B$	B	B	$a + b$
034	+	40	E	B	B	B	$a + b$
035	RCL \times 0	45,20, 0	$E(a + b)$	B	B	B	$a + b$
036	π	43 26	π	$E(a + b)$	B	B	$a + b$
037	\times	20	P	B	B	B	$a + b$
038	RTN	43 32					

D: HP-16C Ramanujan approximation

The 16C is intended for calculations with binary integers but it does have a floating-point mode so the Ramanujan approximation can be implemented. Several parts of the program for the 11C must be modified, because:

- The 16C does not have direct register arithmetic instructions.
- The functions x^2 , y^x , and 10^x are not implemented.
- π is not available as a constant.

These omissions can all be overcome:

- x^2 is calculated with ENTER \times .
- y^x is used to find y^5 , which is done as $y \cdot y^2 \cdot y^2$.

- 10^x is just used to enter 10, which can also be done as EEX 1.
- The value for π is stored in register R0 before executing the program.

Enable FLOAT mode on the 16C before entering the code below. Before executing A, store π to R0 with 3.141592654 STO 0. The program uses R0, R1, and all stack levels. π remains in R0 after execution. I only tested a few cases, but all give exactly the same results as the 11C program, unsurprisingly.

Line	Instruction	Keycode	Line	Instruction	Keycode
001	LBL A	43,22, A	Find $B = 3\lambda^{10}/2^{17}$		
	Find $b - a$		026	3	3
002	ENTER	36	027	\times	20
003	R↓	33	028	4	4
004	X<>Y	34	029	/	10
005	—	30	Find $M = \sqrt{4 - L}$		
	Find $a + b$, save in R1		030	X<>Y	34
006	LST x	43 36	031	ENTER	36
007	R↑	43 33	032	ENTER	36
008	+	40	033	4	4
009	STO 1	44 1	034	X<>Y	34
	Find $L = 3\lambda^2$		035	—	30
010	/	10	036	\sqrt{x}	43 25
011	3	3	Find $D = L/(10 + M)$		
012	X<>Y	34	037	EEX	42 49
013	ENTER	36	038	1	1
014	\times	20	039	+	40
015	\times	20	040	/	10
	Find $A = \lambda^{10}/2^{15}$		Find $E = D + B + 1$		
016	LST x	43 36	041	+	40
017	8	8	042	1	1
018	/	10	043	+	40
019	ENTER	36	Find $P = \pi(a + b)E$		
020	\times	20	044	RCL 1	45 1
021	LST x	43 36	045	\times	20
022	X<>Y	34	046	RCL 0	45 0
023	\times	20	047	\times	20
024	LST x	43 36	048	RTN	43 21
025	\times	20			

E: HP-20S Ramanujan approximation

The 20S is not an RPN calculator but is programmable. Storage registers R0, R1, and R2 are used. The checksum is 49E4 if A is the only program in memory. Program A uses 59 of the 99 available program lines.

To find the ellipse perimeter for $a = 4$ and $b = 1$, use

$$4 \text{ [INPUT]} 1 \text{ [XEQ]} A \implies 17.1568414364$$

Line	Code	Keycode	R0	R1	R2
01	LBL A	61 41 A	?	?	?
02	STO 1	21 1	?	b	?
03	SWAP	51 31	?	b	?
04	STO 0	21 0	a	b	?
05	1	1	a	b	?
06	STO 2	21 2	a	b	1
07	(33	a	b	1
08	RCL 0	22 0	a	b	1
09	—	65	a	b	1
10	RCL 1	22 1	a	b	1
11)	34	a	b	1
12	÷	45	a	b	1
13	(33	a	b	1
14	RCL 0	22 0	a	b	1
15	+	75	a	b	1
16	RCL 1	22 1	a	b	1
17)	34	a	b	1
18	STO 0	21 0	$a+b$	b	1
19	=	74	$a+b$	b	1
20	STO 1	21 1	$a+b$	λ	1
21	y^x	14	$a+b$	λ	1
22	1	1	$a+b$	λ	1
23	0	0	$a+b$	λ	1
24	×	55	$a+b$	λ	1
25	3	3	$a+b$	λ	1
26	÷	45	$a+b$	λ	1
27	2	2	$a+b$	λ	1
28	y^x	14	$a+b$	λ	1
29	1	1	$a+b$	λ	1
30	7	7	$a+b$	λ	1
31	=	74	$a+b$	λ	1
32	STO+ 2	21 75 2	$a+b$	λ	$1+c\lambda^{10}$
33	RCL 1	22 1	$a+b$	λ	$1+c\lambda^{10}$
34	x^2	51 11	$a+b$	λ	$1+c\lambda^{10}$
35	×	55	$a+b$	λ	$1+c\lambda^{10}$
36	3	3	$a+b$	λ	$1+c\lambda^{10}$
37	=	74	$a+b$	λ	$1+c\lambda^{10}$
38	STO 1	21 1	$a+b$	$3\lambda^2$	$1+c\lambda^{10}$
39	/	45	$a+b$	$3\lambda^2$	$1+c\lambda^{10}$
40	(33	$a+b$	$3\lambda^2$	$1+c\lambda^{10}$
41	(33	$a+b$	$3\lambda^2$	$1+c\lambda^{10}$
42	4	4	$a+b$	$3\lambda^2$	$1+c\lambda^{10}$

Line	Code	Keycode	R0	R1	R2
43	—	65	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
44	RCL 1	22 1	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
45)	34	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
46	\sqrt{x}	11	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
47	+	75	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
48	1	1	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
49	0	0	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
50)	34	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
51	+	75	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
52	RCL 2	22 2	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
53	=	74	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
54	×	55	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
55	RCL 0	22 0	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
56	×	55	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
57	π	61 22	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
58	=	74	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$
59	RTN	61 26	$a + b$	$3\lambda^2$	$1 + c\lambda^{10}$

F: HP-28 Ramanujan approximation

Program 'EPRA' listed below runs on the HP-28C and 28S calculators. Flag 35 should be *clear* for numeric results, otherwise π remains in the returned result. For the test cases in Appendix M, the maximum error is about 6 digits in the last place.

EPRA	(about 118 bytes)
Code	Comment
«	
DUP2 + ROT ROT - OVER /	Find λ and $a + b$
DUP 10 \wedge 2.28881835938E-5 *	Find $c\lambda^{10}$
SWAP SQ 3 *	Find $L = 3\lambda^2$
DUP 4 SWAP - $\sqrt{\quad}$	Find $M = \sqrt{4 - L}$
10 + /	Find $Q = L/(10 + M)$
1 + +	Find $1 + Q + c\lambda^{10}$
* π *	Find $P = \pi(a + b)(1 + Q + c\lambda^{10})$
»	

G: HP 32s Ramanujan approximation

This program for the Ramanujan approximation program works with the various versions of the HP 32S; I used a 32SII for testing. Storage register Z is used, as well as all stack levels. The program uses 63.5 bytes.

Line	Code	x	y	z	t	Comment
A01	LBL A	b	a	?	?	Find $b - a$
A02	ENTER	b	b	a	?	
A03	R↓	b	a	?	b	
A04	X<>Y	a	b	?	b	
A05	—	$b - a$?	b	b	
A06	LAST x	a	$b - a$?	b	Find $b + a$
A07	R↑	b	a	$b - a$?	
A08	+	$b + a$	$b - a$?	?	
A09	STO Z	$b + a$	$b - a$?	?	Save $b + a$
A10	÷	λ	?	?	?	Find λ
A11	3	3	λ	?	?	Find $L = 3\lambda^2$
A12	X<>Y	λ	3	?	?	
A13	x^2	λ^2	3	?	?	
A14	×	L	?	?	?	
A15	LAST x	λ^2	L	?	?	Find $c\lambda^{10}$
A16	5	5	λ^2	L	?	
A17	y^x	λ^{10}	L	?	?	
A18	c^*	c^*	λ^{10}	L	?	
A19	×	R	L	?	?	$R = c\lambda^{10}$
A20	X<>Y	L	R	?	?	$M = \sqrt{4 - L}$
A21	4	4	L	R	?	
A22	X<>Y	L	4	R	?	
A23	—	$4 - L$	R	?	?	
A24	LAST x	L	$4 - L$	R	?	
A25	X<>Y	$4 - L$	L	R	?	
A26	SQRT	M	L	R	?	
A27	10	10	M	L	R	
A28	+	$M + 10$	L	R	R	
A29	÷	Q	R	R	R	$Q = L/(M + 10)$
A30	+	$Q + R$	R	R	R	
A31	1	1	$Q + R$	R	R	
A32	+	S	R	R	R	$S = 1 + Q + R$
A33	RCL Z	$a + b$	S	R	R	
A34	×	$S(a + b)$	R	R	R	
A35	π	π	$S(a + b)$	R	R	
A36	×	P	R	R	R	
A37	RTN					

* $c = 2.28881835938E-5$

H: HP 33s, 35s Ramanujan approximation

The program above for the HP 32 series could be used for the HP 35s, but we can use some direct register arithmetic instructions to save a few bytes, since we need to use a storage register, anyway. All the stack levels and register Z are used. For the 35s program size is 121 bytes and the checksum (for what it's worth) is 246A. This program also works with the 33s, with program size 171 and checksum 1A99.

Line	Code	x	y	z	t	Z	Comment
A001	LBL A	b	a	?	?	?	
A002	STO Z	b	a	?	?	b	
A003	X<>Y	a	b	?	?	b	
A004	STO- Z	a	b	?	?	$b - a$	
A005	+	$a + b$?	?	?	$b - a$	
A006	STO÷ Z	$a + b$?	?	?	$-\lambda$	$-\lambda =$
A007	X<> Z	$-\lambda$?	?	?	$a + b$	$(b - a)/(a + b)$
A008	3	3	$-\lambda$?	?	$a + b$	
A009	X<>Y	$-\lambda$	3	?	?	$a + b$	
A010	x^2	λ^2	3	?	?	$a + b$	
A011	\times	L	?	?	?	$a + b$	$L = 3\lambda^2$
A012	LAST x	λ^2	L	?	?	$a + b$	
A013	5	5	λ^2	L	?	$a + b$	
A014	y^x	λ^{10}	L	?	?	$a + b$	
A015	c	c^*	λ^{10}	L	?	$a + b$	(see below)
A016	\times	R	L	?	?	$a + b$	
A017	X<>Y	L	R	?	?	$a + b$	$R = c\lambda^{10}$
A018	4	4	L	R	?	$a + b$	
A019	X<>Y	L	4	R	?	$a + b$	
A020	-	$4 - L$	R	?	?	$a + b$	
A021	LAST x	L	$4 - L$	R	?	$a + b$	
A022	X<>Y	$4 - L$	L	R	?	$a + b$	
A023	\sqrt{x}	M	L	R	?	$a + b$	$M = \sqrt{4 - L}$
A024	10	10	M	L	R	$a + b$	
A025	+	$M + 10$	L	R	R	$a + b$	
A026	÷	Q	R	R	R	$a + b$	$Q = L/(10 + M)$
A027	+	$Q + R$	R	R	R	$a + b$	
A028	1	1	$Q + R$	R	R	$a + b$	
A029	+	S	R	R	R	$a + b$	$S = Q + R + 1$
A030	STO \times Z	S	R	R	R	$S(a + b)$	
A031	π	π	S	R	R	$S(a + b)$	
A032	RCL \times Z	P	π	S	R	$S(a + b)$	$P = \pi(a + b)S$
A033	RTN						

* $c = 2.288\ 8183\ 5939E-5$

I: HP 39g Ramanujan approximation

(This program is more of a curiosity than anything else. This is the first program I've written for the 39g and also likely the last. All I've got for documentation is the User's Guide, which is a little slim on actual programming details: for example, how to return a program result to the history stack? how to take program arguments from the history stack? and so on.)

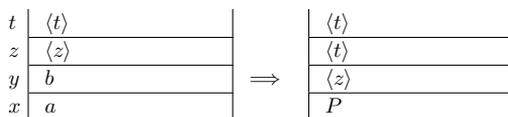
Program EPR implements Ramanujan's approximation. EPR prompts for the two semi-axes A and B, then displays the perimeter on the graphics screen. The perimeter is also saved in real variable P. EPR uses real variables C and D. I only tested a few values for a and b ; they matched the 50g results to within one or two units in the last place. The calculation is essentially

the same as the 50g version, but converted into algebraic. The perimeter can be retrieved after the program runs from variable P.

EPR	(0.16 KB)
PROMPT A: PROMPT B:	Prompt for (and save) axes A and B
(A-B)/(A+B)► C:	Find C = λ
3*C ² ► D:	Find D = $3\lambda^2$
(2.28881835938E-5*C ¹⁰ +	Find $3\lambda^{10}/2^{17}$
D/(10+ $\sqrt{4-D}$)+ 1)	Finish perimeter calculation ...
(A+B) π ► P:	... and save perimeter in P
ERASE:	Clear the graphics screen
DISP 1; "ELLIPSE PERIMETER=":	Display label on display line 1 ...
DISP 2; P:	... and the perimeter on line 2
FREEZE	Freeze the graphics display

J: HP 42s Ramanujan approximation

With the HP-42s we have some memory and a more sophisticated instruction set, compared to some other calculators in this paper, so we'll use those features to write a slightly different ellipse perimeter program. It is usually preferable that function evaluation programs mimic the built-in functions such that storage registers are undisturbed and the stack registers are restored. For example, the ellipse perimeter program has two input arguments a and b and a single result P , so we would like the stack I/O diagram to look like



where the notation $\langle R \rangle$ means 'the original contents of register R'. We could build the code for this into the ellipse perimeter program, but memory is limited on the 42s. We may well have other function programs which need the same save and restore functionality, so I wrote two programs to that end. Program SZT0 saves stack levels z and t and register R00 in named variables. Program RZT0 restores the saved values and deletes the named variables. Of course this feature increases code size and execution time, so it becomes a matter of your priorities and preferences. The total required memory to use EPR is 149 bytes, but STZ0 and RTZ0 can be used with other programs.

Code listings for SZT0, RZT0, and ERP are given below. ERP calculates the ellipse perimeter, with calls to SZT0 and RZT0 at lines 2 and 32, respectively.

SZT0 — Save stack levels z , t , and R00 (28 bytes)					
Line	Code	x	y	z	t
01	LBL "STZ0"	$\langle x \rangle$	$\langle y \rangle$	$\langle z \rangle$	$\langle t \rangle$
02	R↑	$\langle t \rangle$	$\langle x \rangle$	$\langle y \rangle$	$\langle z \rangle$
03	STO "0T"	$\langle t \rangle$	$\langle x \rangle$	$\langle y \rangle$	$\langle z \rangle$
04	R↓	$\langle y \rangle$	$\langle z \rangle$	$\langle x \rangle$	$\langle x \rangle$
05	STO "0Z"	$\langle z \rangle$	$\langle t \rangle$	$\langle x \rangle$	$\langle y \rangle$
06	R↓	$\langle t \rangle$	$\langle x \rangle$	$\langle y \rangle$	$\langle z \rangle$
07	RCL 00	$\langle R00 \rangle$	$\langle t \rangle$	$\langle x \rangle$	$\langle y \rangle$
08	STO "0R0"	$\langle R00 \rangle$	$\langle t \rangle$	$\langle x \rangle$	$\langle y \rangle$
09	R↑	$\langle y \rangle$	$\langle R00 \rangle$	$\langle t \rangle$	$\langle x \rangle$
10	R↑	$\langle x \rangle$	$\langle y \rangle$	$\langle R00 \rangle$	$\langle t \rangle$
11	RTN	$\langle x \rangle$	$\langle y \rangle$	$\langle R00 \rangle$	$\langle t \rangle$

RZT0 — Restore stack levels z , t , and R00 (42 bytes)					
Line	Code	x	y	z	t
01	LBL "RTZ0"	$\langle x \rangle$	$\langle y \rangle$	$\langle z \rangle$	$\langle t \rangle$
02	RCL "0R0"	$\langle 0R0 \rangle$	$\langle x \rangle$	$\langle y \rangle$	$\langle z \rangle$
03	STO 00	$\langle 0R0 \rangle$	$\langle x \rangle$	$\langle y \rangle$	$\langle z \rangle$
04	R↓	$\langle x \rangle$	$\langle y \rangle$	$\langle z \rangle$	$\langle 0R0 \rangle$
05	RCL "0T"	$\langle 0T \rangle$	$\langle x \rangle$	$\langle y \rangle$	$\langle z \rangle$
06	RCL "0T"	$\langle 0T \rangle$	$\langle 0T \rangle$	$\langle x \rangle$	$\langle y \rangle$
07	RCL "0Z"	$\langle 0Z \rangle$	$\langle 0T \rangle$	$\langle 0T \rangle$	$\langle x \rangle$
08	R↑	$\langle x \rangle$	$\langle 0Z \rangle$	$\langle 0T \rangle$	$\langle 0T \rangle$
09	CLV "0T"	$\langle x \rangle$	$\langle 0Z \rangle$	$\langle 0T \rangle$	$\langle 0T \rangle$
10	CLV "0Z"	$\langle x \rangle$	$\langle 0Z \rangle$	$\langle 0T \rangle$	$\langle 0T \rangle$
11	CLV "0R0"	$\langle x \rangle$	$\langle 0Z \rangle$	$\langle 0T \rangle$	$\langle 0T \rangle$
12	RTN				

EPR — Ramanujan Ellipse Approximation (79 bytes)

Line	Code	Line	Code	Line	Code
01	LBL "EPR"	14	×	27	c^*
02	XEQ "SZT0"	15	4	28	×
03	RCL ST Y	16	$X \langle \rangle Y$	29	+
04	$X \langle \rangle Y$	17	—	30	1
05	—	18	LAST $_x$	31	+
06	LAST $_x$	19	$X \langle \rangle Y$	32	STO × 00
07	RCL ST Z	20	SQRT	33	RCL 00
08	+	21	10	34	PI
09	STO 00	22	+	35	×
10	÷	23	÷	36	XEQ "RZT0"
11	$X \uparrow 2$	24	$X \langle \rangle Y$	37	RTN
12	STO ST Y	25	5		
13	3	26	$Y \uparrow X$		

* $c = 2.288\ 8183\ 5938\ E-5$

K: HP 48 series Ramanujan approximation

The 50g program ELPRA is used as a starting point for this program EPR for the HP 48 series calculators; I used a 48GX. The 48 series calculators lack the UNROT command, so ROT ROT is used instead. System flag -2 should be *set* for numeric results; otherwise π remains in the result.

EPR	(121.5 bytes, checksum#5BA0)
«	
DUP2 -	Find $a - b$
ROT ROT +	Find $a + b$
DUP π *	Find $k = \pi(a + b)$
ROT ROT /	Find $\lambda = (a - b)/(a + b)$
DUP SQ 3 * DUP	Find $L = 3\lambda^2$
4 SWAP - $\sqrt{\quad}$	Find $M = \sqrt{4 - L}$
10 + / 1 +	Find $Q + 1 = L/(10 + M)$
SWAP $10 \wedge 2.28881835938E-5$ *	Find $c\lambda^{10}$
+ *	Find $P = k(1 + Q + c\lambda^{10})$
»	

L: HP 95LX Ramanujan approximation

The 95LX is a handheld computer, or pocket computer, and unfortunately does not have any programming languages built-in. It does have a version of the Lotus 123 spreadsheet, which can be used to calculate the Ramanujan approximation. The table below specifies the cell entries.

Cell	Cell contents
A1	'a=
A2	'b=
B1	<i>semi-major axis a</i>
B2	<i>semi-major axis b</i>
C1	'lambda: (optional)
C2	'3*lam^2 (optional)
D1	$(B1 - B2)/(B1 + B2)$
D2	$3 * D1^2$
A4	'Perimeter:
A5	$@PI * (B1 + B2) * (1 + D2 / (10 + @SQRT(4 - D2))) + 3 * D1^10 / 2^17$

C1 and C2 are labels for D1 and D2 and need not be entered. Semi-axes a and b are entered in B1 and B2, with the perimeter shown in cell A5.

M: Ellipse perimeter test cases

The table below lists some test values for a and b which may be used to test programs for calculating the Ramanujan approximation P_R . The value for λ is also given, as well as the actual value of the perimeter P_{ACT} .

a	b	$\lambda = (a - b)/(a + b)$	P_R	P_{ACT}
1.0	1.0	0	6.2831 85307 17959	6.2831 85307 17959
1.25	1.0	0.11111 11111 11111	7.0904 16972 24362	7.0904 16972 24362
1.5	1.0	0.20000 00000 00000	7.9327 19794 64402	7.9327 19794 64529
1.9	1.0	0.31034 48275 86207	9.3313 42102 01233	9.3313 42102 31999
2.3	1.0	0.39393 93939 39394	10.773 53733 57045	10.773 53734 23153
3.0	1.0	0.50000 00000 00000	13.364 89306 07070	13.364 89322 05553
4.0	1.0	0.60000 00000 00000	17.156 84143 63427	17.156 84355 03137
5.7	1.0	0.70149 25373 13433	23.730 07592 40255	23.730 09935 88423
9.0	1.0	0.80000 00000 00000	36.687 58358 21579	36.687 81772 76102
19.0	1.0	0.90000 00000 00000	76.400 42095 25457	76.403 59077 05161
39.0	1.0	0.95000 00000 00000	156.21 56750 26899	156.23 33761 19845
199.0	1.0	0.99000 00000 00000	795.83 31302 82852	796.06 21070 57088
10^3	1.0	0.99800 19980 01998	3998.5 72446 23787	4000.0 15588 10469
10^4	1.0	0.99980 00199 98000	39984. 72878 48971	40000. 00201 93270

N: References

- [1] *NIST Handbook of Mathematical Functions*, Frank W. J. Olver et al, 2010, Cambridge University Press. See section 19.9, p494. Also available at dlmf.nist.gov.
- [2] *Gauss, Landen, Ramanujan, the Arithmetic-Geometric Mean, Ellipses, π , and the Ladies Diary*, Gert Alvkvist and Bruce Berndt, 1988, The American Mathematical Monthly, Vol. 95 No. 7, Mathematical Association of America. I found a copy at archive.org, which is down as of this writing. You may be able to find the article with the doi: 10.2307/2323302.
- [3] *Handbook of Mathematics and Computational Science*, John W. Harris and Horst Stoeckel, 1998, Springer-Verlag. For general information about ellipses you could certainly check Wikipedia and other online sources. For a printed reference, this handbook is a good choice. Section 3.8.7 defines the major and minor semi-axes, gives the exact expression the ellipse area, and a very crude estimation for the perimeter:

$$P \approx \pi \left(1.5(a + b) + \sqrt{ab} \right)$$

Section 8.4 gives most of the common ellipse properties and relations, including equations in various forms and coordinate systems, diameters, tangents, curvature, and equations for areas of a segments and a sector. A better estimate of the complete perimeter is given as

$$P \approx \pi(a + b) \frac{64 - 3\lambda^4}{64 - 16\lambda^2}, \quad \lambda = \frac{a - b}{a + b}$$

- [4] *Perimeter of an Ellipse, Final Answers*, Gérard P. Michon, 2023-08-19, <https://www.numericana.com/answer/ellipse.htm#elliptic>. Describes almost two dozen ellipse perimeter approximations; oddly does not appear to include Ramanujan's better approximation used in this paper.

- [5] *Srinivasa Ayengar Ramanujan*, Roy Porter and Marilyn Ogilvie, eds., 2000, The Biographical Dictionary of Scientists. This entry is a short summary of Ramanujan's life and work.
- [6] *The Man Who Knew Infinity*, Robert Kanigel, 1991, Charles Scribner's Sons. Kanigel's book is a detailed popular biography of Ramanujan's life and work. Ramanujan's mathematics is described in a little more detail than in most books like this, and Kanigel successfully explains why Ramanujan's work is so important.
- [7] *Srinivasa Aiyengar Ramanujan*, J. J. O'Conner and E. F. Robertson, 1998, the MacTutor website (<https://mathshistory.st-andrews.ac.uk/Biographies/Ramanujan/>). A shorter read than Kanigel's book, but covers most of the high points. Includes 32 additional references.
- [8] *Chebyshev Approximations for the Complete Elliptic Integrals K and E* , W. J. Cody, 1965, Mathematics of Computation v19, pp. 105–112. This table describes the process used to calculate the coefficients of the Chebyshev polynomial approximations for orders 2 through 10. The maximum absolute approximation errors for the approximations are listed in table Ia. The lower-order approximations may be candidates for use with calculators without as much memory as the 50g. Note that a subsequent corrigenda corrects an error in Table III: the value of c_7 for $n = 8$ should be 7.33561 64974 29036 5 (−03).