π auf 10 000 000 (und mehr) Dezimalen genau berechnen

Ac 2011-2021

Wegen **π = 4 · arctan(1)** kann die Reihenentwicklung des Arkustangens verwendet werden.

$$\arctan(x) = \sum_{k=0}^{\infty} (-1)^k \cdot \frac{x^{2k+1}}{2k+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \frac{x^{11}}{11} + -\ldots + (-1)^n \cdot \frac{x^{2n+1}}{2n+1} \pm \ldots \; ; \quad |x| \le 1$$

Eine Darstellung für $\pi / 4$ wäre dann also (Leibniz-Reihe):

$$\frac{\pi}{4} = \arctan(1) = \sum_{k=0}^{\infty} (-1)^k \cdot \frac{1}{2k+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + -\ldots + (-1)^n \cdot \frac{1}{2n+1} \pm \ldots$$

Die Leibniz-Reihe konvergiert aber sehr schlecht, wie folgendes Beispiel zeigt:
Eine Berechnung von **π** mittels **4 · arctan(1)** auf lediglich 5 Dezimalen unterliegt der Bedingung :

Restglied $| R_n | < 5 \cdot 10^{-6}$, d.h. $1/(4n) < 5 \cdot 10^{-6}$ bzw. $4 \cdot 1/(4n) < 5 \cdot 10^{-6}$, d.h. $n > 2 \cdot 10^5$, somit n > 200 000 .

Es sind also ca. 200 000 Glieder der Reihe für eine 5-stellige Genauigkeit erforderlich.
Die folgende Rechnung belegt dies !

$$4 \cdot \sum_{k=0}^{200000} (-1)^k \cdot \frac{1}{2k+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + -\ldots + (-1)^{200000} \cdot \frac{1}{400001} = 3,14159765\ldots$$

Wegen der schlechten Konvergenz verwendet man besser Kombinationen mehrerer arctan-Reihen.

Beispiele für solche arctan-Kombinationen :

π = 16·arctan(1/5) - 4·arctan(1/239)	**Machin 1706**
π = 48·arctan(1/18) + 32·arctan(1/57) – 20·arctan(1/239)	**Gauß**
π = 176·arctan(1/57) + 28·arctan(1/239) - 48·arctan(1/682) + 96·arctan(1/12943)	**Störmer 1896**

Anmerkung:
Die STÖRMER-Reihe konvergiert von diesen 3 Reihen am besten, wie man am “PerformanceIndex PI” sieht. Dabei ist PI folgendermaßen definiert: PI = 1 / lg(z1) + 1 / lg(z2) + 1 / lg(z3) + …

Machin: PI = 1 / lg(5) + 1 / lg(239) = 1,85
Gauß: PI = 1 / lg(18) + 1 / lg(57) + 1 / lg(239) = 1,79
Störmer: PI = 1 / lg(57) + 1 / lg(239) + 1 / lg(682) + 1 / lg(12943) = 1,59

In allen diesen Formeln wird arctan(1/z) (z ganzzahlig und positiv) verwendet:

$$\arctan(\frac{1}{z}) = \sum_{k=0}^{\infty} (-1)^k \cdot (\frac{1}{z})^{2k+1} / (2k+1) =$$

$$\frac{1}{z} - (\frac{1}{z})^3 / 3 + (\frac{1}{z})^5 / 5 - (\frac{1}{z})^7 / 7 + - \ldots + (-1)^n \cdot (\frac{1}{z})^{2n+1} / (2n+1) \pm \ldots$$

Wir betrachten im folgenden die MACHIN-Reihe (und nicht die besser konvergierende Störmer-Reihe), weil sie aus lediglich 2 arctan-Gliedern besteht und vergleichsweise gut konvergiert.

Anmerkung: **John Machin** berechnete 1706 mit seine Reihe (handschriftlich !) 100 Dezimalen von π .

Zu berechnen sind:

$16 \cdot \arctan(1 / 5) = 16 \cdot [1 / 5 - (1 / 5)^3 / 3 + (1 / 5)^5 / 5 - (1 / 5)^7 / 7 + - \ldots]$ sowie
$4 \cdot \arctan(1 / 239) = 4 \cdot [1 / 239 - (1 / 239)^3 / 3 + (1 / 239)^5 / 5 - (1 / 239)^7 / 7 + - \ldots]$

Ersichtlich ist hier keine Multiplikation erforderlich, sondern nur Addition, Subtraktion, Division .
Die Multiplikationen mit 16 und 4 lassen sich durch Additionen ersetzen !

Zunächst müssen wir das für jede der beiden arctan-Reihen benötigte minimale n (bzw. 2n+1) bestimmen:

Es sei **nk** die Anzahl der gewünschten korrekten Nachkommastellen (Dezimalen).
Verändert nun der n-te Summand der Reihe, multipliziert mit dem jeweiligen arctanfaktor f (16 bzw. 4), die nk-te Dezimale nicht mehr, so ist die gewünschte Genauigkeit erreicht.

Dies ist dann der Fall, wenn $f \cdot (1/z)^{2n+1} / (4n)$ kleiner ist als $0{,}5 \cdot 10^{-nk}$.

Ansatz also: $\mathbf{f \cdot (1/z)^{2n+1} / (4n) < 0{,}5 \cdot 10^{-nk}}$ **mit z > 1 ; n ∈ IN**

Umformungen:

$(1/z)^{2n+1} / n < 2 / f \cdot 10^{-nk}$
$\lg((1/z)^{2n+1} / n) < \lg(2 / f \cdot 10^{-nk})$
$\lg((1/z)^{2n+1}) - \lg(n) < \lg(2) - \lg(f) + \lg(10^{-nk})$
$(2n+1) \cdot \lg(1/z) - \lg(n) < \lg(2) - \lg(f) - nk$
$(2n+1) \cdot (-\lg(z)) \leq \lg(2) - \lg(f) - nk$
$2n+1 \geq [\, nk + \lg(f / 2) \,] / \lg(z)$

$n \geq [\, (nk + \lg(f / 2)) / \lg(z) - 1] / 2$

Beispiel: nk = 100 ; z = 5 ; f = 16 : n ≥ 72 nk = 100 ; z = 239 ; f = 4 : n ≥ 72

n gegeben ; nk = ?

$(1/z)^{2n+1} / n < 2 / f \cdot 10^{-nk}$
$f / 2 \cdot (1/z)^{2n+1} / n < 10^{-nk}$
$\lg (f / 2 \cdot (1/z)^{2n+1} / n) < - nk$
$nk < -\lg (f / 2 \cdot (1/z)^{2n+1} / n)$
$nk < \lg (2 \cdot n / f \cdot z^{2n+1})$

$nk < \lg (2 \cdot n / f) + (2 \cdot n+1) \cdot \lg(z)$

Beispiel: n = 100 ; z = 5 ; f = 16 : nk < 141,5 n = 100 ; z = 239 ; f = 4 : nk < 487,8

Beispiele (In der Praxis verwendet man noch „Schutzstellen“, so dass nk z.B. zu **nk+10** wird !) :

nk	**nk+10**	**2n+1 für 16·arctan(1/5)**	**2n+1 für 4·arctan(1/239)**
$100 = 10^2$	110	161	47
$1000 = 10^3$	1010	1 447	425
$10000 = 10^4$	10010	14 323	4 209
$100000 = 10^5$	100010	143 085	42 051
$1000000 = 10^6$	1000010	1 430 693	420 457
$6500000 = 6{,}5 \cdot 10^6$	6500010	9 299 415	2 732 937
$10000000 = 10^7$	10000010	14 306 783	4 204 517

Näherungswerte:

16·arctan(1/5) = 3,1583289575980924

4·arctan(1/239) = 0.016736304008298893

Die Differenz 3.1415926535897936 ist dann ein Näherungswert für die Kreiszahl π .

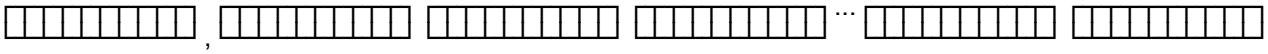
Hinweise zu den verwendeten Datenstrukturen:

Zur Berechnung der Nachkommastellen von π wird das **10 000 000 000 - System** verwendet, d.h. anstelle der Basis 10 (Ziffern 0 bis 9) verwenden wir nun die Basis 10 000 000 000 = 10^{10} , diese beinhaltet die 10^{10} Ziffern 0 bis 9 999 999 999 .

Die Ziffern zur Basis 10 000 000 000 werden in einem Feld (Array) namens x abgespeichert. Wünscht man z.B. 10 000 Dezimalen so bedeutet dies ein Feld, welches von x[0] bis zu x[1000] reicht, denn jedes Element (Zelle) des Feldes kann 10 Ziffern aufnehmen.

Achtung: **Die Vorkommastelle 3 von π wird stets in x[0] abgelegt, die Dezimalen in x[1] bis x[n]** .

Grafische Veranschaulichung des Feldes x mit n+1 Feldelementen (Registern) je 10 Ziffern:



Die **Grundrechenarten** + - / * für das Rechnen im 10 000 000 000 – System :

Anmerkung: Die Multiplikation wird zwar für die Approximation von Pi hier nicht benötigt, sie kann aber bei anderen Rechnungen mit RegisterArithmetik verwendet werden.

*Beispiel für die **Summenbildung** mit 3 Registern der Länge 10:*

Zahl	x[0]	,	x[1]	x[2]
a	0000006053	,	3001528335	2970071009
b	0000000247	,	9459107378	8450300256
c	0000006301	,	2460635714	1420371265

Erläuterung: Die **Summe c = a + b** wird "von rechts nach links mit Übertrag" wie folgt gebildet.

1. x[2] von a + x[2] von b = x[2] von c = 1420371265 ; Übertrag = 1
2. x[1] von a + x[1] von b + Übertrag = x[1] von c = 2460635714; Übertrag = 1
3. x[0] von a + x[0] von b + Übertrag = x[0] von c = 6301 ; Übertrag = 0

Bei der Pi-Berechnung kann ein Übertrag von x[0] **nicht** vorkommen, denn x[0] hat immer den Wert 3 !

Die Zeichenkette x[0] , x[1] x[2] stellt die Summe c dar : c = 6301,24606357141420371265

*Beispiel für die **Differenzbildung** mit 3 Registern der Länge 10:*

Zahl	x[0]	,	x[1]	x[2]
a	0000006053	,	3001528335	2970071009
b	0000000247	,	9459107378	8450300256
c	0000005805	,	3542420956	4519770753

Erläuterung: Die **Differenz c = a - b** wird "von rechts nach links mit Übertrag" wie folgt gebildet.

1. x[2] von a - x[2] von b = x[2] von c = -5480229247 < 0

Falls Differenz negativ, dann setze Differenz = Differenz + Basis und setze Übertrag = 1

also x[2] von c = 4519770753 ; Übertrag = 1

2. x[1] von a - x[1] von b - Übertrag = x[1] von c = -6457579044 < 0

also x[1] von c = 3542420956 ; Übertrag = 1

3. x[0] von a - x[0] von b - Übertrag = x[0] von c = 0000005805 ; Übertrag = 0

Die Zeichenkette x[0] , x[1] x[2] stellt die Differenz c dar : c = 5805,35424209564519770753

*Beispiel für die **Quotientenbildung** mit **5** Registern der Länge 10 (Beispiel aus der Machin-Reihe):*

Anmerkung:
Die Quotienten bestehen hier lediglich aus der **Division einer RegisterZahl durch eine LongZahl** !
Dies reicht für die Machin-Reihe aus, weil nur Quotienten vom Typ $1 / z^k$ benötigt werden.

Zahl	x[0] ,	x[1]	x[2]	x[3]	x[4]
a	0000000001 ,	0000000000	0000000000	0000000000	0000000000
b	239 (Datentyp long , kein Feld von Registern !)				
c	0000000000 ,	0041841004	1841004184	1004184100	4184100418

Erläuterung: Der **Quotient c = a / b** wird "von links nach rechts mit Übertrag" wie folgt gebildet.
Hinweis: In obigem Beispiel ist der Spezialfall der Kehrwertbildung von 239 dargestellt !

1. tmp = x[0] + Übertrag*Basis = 1
   tmp div b = = x[0] von c = **0**
   tmp mod b = Übertrag = 1

2. tmp = x[1] + Übertrag*Basis = 10000000000
   tmp div b = = x[1] von c = 41841004
   Nullen einfügen ! x[1] von c = **0041841004**
   tmp mod b = Übertrag = 44

3. tmp = x[2] + Übertrag*Basis = 440000000000 (long-Datentyp)
   tmp div b = = x[2] von c = **1841004184**
   tmp mod b = Übertrag = 24

4. tmp = x[3] + Übertrag*Basis = 240000000000 (long-Datentyp)
   tmp div b = = x[2] von c = **1004184100**
   tmp mod b = Übertrag = 100

5. tmp = x[4] + Übertrag*Basis = 1000000000000 (long-Datentyp)
   tmp div b = = x[2] von c = **4184100418**
   tmp mod b = Übertrag = 98

Die Zeichenkette x[0] , x[1] x[2] x[3] x[4] stellt den Quotienten c dar :
c = 0 , 00418410041841004184100418410041841004184100418

Hier wird übrigens klar, dass wir mindestens ein zusätzliches Register mitführen müssen, damit Rundungsfehler bzw. Divisionsreste aufgefangen werden.

*Beispiel für die **Produktbildung** (RegisterZahl * LongZahl !) mit 3 Registern der Länge 10:*

Zahl	x[0] ,	x[1]	x[2]
a	0000006053 ,	3001528335	2970071009
b	0000073247 ,	0000000000	0000000000
c	0443386076 ,	2945975499	8791196223

Erläuterung: Das **Produkt c = a * b** wird wie folgt gebildet.

1. x[2] von a * x[0] von b = x[2] von c = 217548791196223
Falls x[2] von c >= 10 000 000 000 , dann setze x[2] von c = x[2] von c mod 10 000 000 000 und setze Übertrag = x[2] von c div 10 000 000 000, also x[2] von c = 8791196223 ; Übertrag = 21754

2. x[1] von a * x[0] von b + Übertrag = x[1] von c = 219852945975499
Falls x[1] von c >= 10 000 000 000 , dann setze x[1] von c = x[1] von c mod 10 000 000 000 und setze Übertrag = x[1] von c div 10 000 000 000, also x[1] von c = 2945975499 ; Übertrag = 21985

3. x[0] von a * x[0] von b + Übertrag = x[0] von c = 443386076
Achtung: Das Register c[0] muss kleiner als die Basis 10 000 000 000, denn links vom Register 0 gibt es kein weiteres Register mehr !

Die Zeichenkette x[0] , x[1] x[2] stellt das Produkt c dar : c = 443386076,29459754998791196223

Algorithmen zu den Grundrechenarten (mit beliebiger Genauigkeit):

Gerechnet wird im System mit der Basis 10 000 000 000 (Ziffern von 0 bis 9 999 999 999).
Somit bearbeiten wir immer 10-stellige Zahlen, die in Registern abgespeichert werden.
Wegen der ungenauen Division verwenden wir noch 2 zusätzliche Register (Schutzstellen).
Für eine gewünschte Genauigkeit mit nk Dezimalen benötigen wir daher außer dem Register x[0] noch weitere nk div 10 + 2 Register. nk sollte möglichst durch 10 teilbar sein .

Zur Berechnung der Teilterme der ArkusTangensSummen benötigen wir Registerfelder der Form

long[ ] reg = new long [n+1] mit n = nk div 10 + 2 .

Konkret werden folgende 4 Registerfelder benötigt:

regSum5 für die Berechnung des arctan(1/5)
regSum239 für die Berechnung des arctan(1/239)
regXPot für die Aufnahme der jeweiligen x-Potenz der ArkusTangensSumme.
regXdurchK für die Aufnahme des jeweiligen $x^{2k+1}/(2k+1)$-Wertes der ArkusTangensSumme.

Für eine Berechnung von Pi auf 10^7 Dezimalen werden somit ca. 4 000 000 Register vom Datentyp **long** (je 64 Bits bzw. 8 Bytes) benötigt. Das ist ein RAM-Bedarf von 32 MB für die Register.

Die Berechnung der ArkusTangensSummen beginnt mit dem jeweiligen x-Wert.

Für die erste Summe ist das x=1/5. Die Summe hat also den Anfangswert 1/5.
In einer k-Schleife kann man dann bei jedem Umlauf durch 25 dividieren, denn $x^3 = (1/5)^3 = 1/5/25$ und $x^5 = (1/5)^5 = 1/5/25/25$ usw. .
Jede Potenz von (1/5) wird in der k-Schleife noch durch (2k+1) dividiert .
Abwechselnd werden die Potenzterme zur aktuellen Summe addiert oder von ihr subtrahiert.

Die zweite Summe mit x=1/239 wird entsprechend behandelt.

Der Näherungswert für Pi ist dann 4·(4·Summe1 - Summe2) .

Anmerkung:

Da ein einzelnes Register vom Java-Datentyp long ist, darf jegliches Operationsergebnis die Zahl longmax = $2^{63} - 1$ = 9223372036854775807 = $9{,}223372036854775807 \cdot 10^{18}$ nicht überschreiten.
Insbesondere muss bei der Division darauf geachtet werden, dass der Term
dividend[i] + uebertrag·cBasis unter diesem Wert bleibt .

Da dividend[i] < cBasis und cBasis = 10 000 000 000 , gilt:
dividend[i] + uebertrag·cBasis ≤ 9999999999 + uebertrag·10000000000 ≤ 9223372036854775807 .
Daraus folgt als Bedingung für den Übertrag: **uebertrag ≤ 922337202** .

Allgemein gilt: uebertrag ≤ (9223372036854775807-(cBasis-1)) / cBasis

Beispiel: cBasis = 10^{14} , dann muss uebertrag ≤ 92232 sein .

Der uebertrag tritt zum Beispiel bei der Division durch den Term (2n+1) auf . Falls 10^8 Dezimalen berechnet werden sollen, so ist (2n+1) höchstens gleich 143 067 673 (siehe Tabelle).
Dies liegt noch unterhalb des Wertes 922 337 203 .

Deswegen sollten mit der gewählten Datenstruktur fehlerfrei $\mathbf{10^8}$ = 100 Millionen Dezimalen berechnet werden können. Die Rechenzeit wäre hierfür allerdings gewaltig !

Hinweis:

Genauere Untersuchungen zeigen, dass man mit dem Java-Datentyp „long“ sogar im 10^{12} - System rechnen kann. Wegen der Übertragsproblematik muss dann jedoch (2n+1) ≤ 9 223 371 sein .

Daher kann man in diesem Fall gemäß obiger Tabelle nur ca. $6{,}5 \cdot 10^6$ Dezimalen berücksichtigen.

Geschwindigkeitssteigerung (Optimierung):

Da die Summanden der Reihe mit steigendem Index k immer kleiner werden, muss man nur solche Register in die Rechnung miteinbeziehen, für deren Index (Registernummer) nr gilt: $(1/z)^k / k > 10^{-10nr}$
Das ist für Nummern **$nr > [lg(z)\cdot k + lg(k)] / 10$** der Fall.

Bei 12-stelligen Registern gilt: $(1/z)^k / k > 10^{-12nr}$ und daher $nr > [lg(z)\cdot k + lg(k)] / 12$.

Die Register mit den Nummern 0 bis nr-1 müssen also nicht neu berechnet werden.
Die Rechenzeit verringert sich durch diese Maßnahme um ca. 40% !

Dies soll am Beispiel z = 5 verdeutlicht werden:

$$\arctan(1/5) = 1/5 - (1/5)^3/3 + (1/5)^5/5 - (1/5)^7/7 +- \ ...$$

Rechnung für 12-stellige Register mit nk = 60 Nachkommastellen (Register Nr. 0 bis 5):

Zu verifizierende Formel: $nr > [lg(5)\cdot k + lg(k)] / 12$

k = 0 bis 5: 0,197397333333 333333333333 333333333333 333333333333 333333333333

k = 0 bis 10: 0,197395559788 975468975468 975468975468 975468975468 975468975468

k = 0 bis 20: 0,197395559849 880761887845 482347030334 646433717641 147981705257

k = 0 bis 30: 0,197395559849 880758370049 514352933183 938633495888 225159962007
hier haben sich nur die Register 2 bis 5 geändert
dies bestätigt die Formel , denn $nr > [lg(5)\cdot 30 + lg(30)] / 12$, also $nr > 1{,}8...$, d.h. $nr \geq 2$

k = 0 bis 40: 0,197395559849 880758370049 765194809996 440719542278 661794301080
hier haben sich nur die Register 3 bis 5 geändert
dies bestätigt die Formel , denn $nr > [lg(5)\cdot 40 + lg(40)] / 12$, also $nr > 2{,}4...$, d.h. $nr \geq 3$

k = 0 bis 50: 0,197395559849 880758370049 765194790293 445948704863 472317093340

k = 0 bis 60: 0,197395559849 880758370049 765194790293 447585103928 790935734812
hier haben sich nur die Register 4 und 5 geändert
dies bestätigt die Formel , denn $nr > [lg(5)\cdot 60 + lg(60)] / 12$, also $nr > 3{,}6...$, d.h. $nr \geq 4$

k = 0 bis 70: 0,197395559849 880758370049 765194790293 447585103787 852089064528

k = 0 bis 80: 0,197395559849 880758370049 765194790293 447585103787 852101517690
hier hat sich nur das Register 5 geändert !
dies bestätigt die Formel , denn $nr > [lg(5)\cdot 80 + lg(80)] / 12$, also $nr > 4{,}8...$, d.h. $nr \geq 5$

k = 0 bis 90: 0,197395559849 880758370049 765194790293 447585103787 852101517688

k = 0 bis 100: 0,197395559849 880758370049 765194790293 447585103787 852101517688
hier hat sich auch das Register 5 nicht mehr geändert !
dies bestätigt die Formel , denn $nr > [lg(5)\cdot 100 + lg(100)] / 12$, also $nr > 5{,}9...$, d.h. $nr \geq 6$

Das Java-Programm

```
import java.util.Arrays;

// Ac 03-2016 bis 01-2021;  Stand: 15.01.2021

public class PiRegArithBasis10hoch12 {

	// Registerarithmetik für die Reihenberechnung der Kreiszahl π
	public final static int cMaxStellen = 6_500_000; // = 6,5·10^6; eventuell auch 10^7 möglich !
	public final static long cBasis = 1_000_000_000_000L; // 10 hoch 12
	public final static int cRegStellen = 12;
	public final static int cZusatzRegister = 2;

	//   Grafische Veranschaulichung der (n+1) Register x mit jeweils 12 Ziffern:
	//
	// [xxxxxxxxxxxx],[xxxxxxxxxxxx][xxxxxxxxxxxx][xxxxxxxxxxxx]···[xxxxxxxxxxxx][xxxxxxxxxxxx]
	//     x[0]          x[1]         x[2]         x[3]            x[n-1]       x[n]
	// Im Register x[0] wird die 3 (ganzzahliger Anteil von π) abgespeichert


	// ================================================================================
	//    Implementierung der Grundrechenarten für die Register
	//    jedes Register ist vom Typ long (nur 12 Ziffern von long werden verwendet)
	// ================================================================================

	// Die folgenden 3 Grundrechenarten sind für die Reihenberechnung von π optimiert:
	// ------------------------------------------------------------------------------------------
	// Anm.:  Auf die Operation "*" kann hier verzichtet werden ( Additionen genügen für zB. "4·" )

	// Die Operation "Quotient" beschränkt sich auf die Division Registerzahl / Longzahl
	// Um Rechenzeit und Speicherplatz zu sparen, wird hier in jeder der Methoden auf die
	// anfängliche Erzeugung einer Zahl "Ergebnis" verzichtet !
	// Stattdessen wird das Ergebnis immer im 1.Operanden (sum1, minuend, ...) gespeichert.
	// Achtung (Seiteneffekt !):  Wegen der dynamischen Datenstruktur der Registerzahlen
	//      wird jedesmal der 1.Operand verändert !!

	public static long[] RegSumme(long[] sum1, long[] sum2, int n, int n0) {
		// n0 = Startregister;              // sum1 = sum1 + sum2
		long uebertrag = 0L;
		for (int i = n; i >= n0; i--) { // Addition von rechts nach links
			sum1[i] = sum1[i] + sum2[i] + uebertrag;
			if (sum1[i] < cBasis)
				uebertrag = 0L;
			else  {
				uebertrag = 1L;
				sum1[i] = sum1[i] - cBasis;
			}
		}
		return sum1;
	}

	public static long[] RegDifferenz(long[] minuend, long[] subtrahend, int n, int n0) {
		// gilt nur für die π-Berechnung, da dort subtrahend[0] = 0 !!
		// minuend = minuend - subtrahend;  Ergebnis >= 0 erforderlich
		for (int i = n; i > n0; i--) { // Subtraktion von rechts nach links
			minuend[i] = minuend[i] - subtrahend[i];
			if (minuend[i] < 0) {
				minuend[i] = minuend[i] + cBasis;
				minuend[i-1]--;
			}
		}
		return minuend;
	}

	public static long[] RegQuotient(long[] dividend, long divisor, int n, int n0) {
		// dividend = dividend / divisor ;  Division RegZahl / LongZahl
		long quot, uebertrag = 0L;
		for (int i = n0; i <= n; i++) { // Division von links nach rechts
			quot = dividend[i] + uebertrag * cBasis;
			dividend[i] = quot / divisor;  // quot div divisor
			uebertrag = quot % divisor;  // quot mod divisor
		}
		return dividend;
	}
```

```
// ====================================
//    Ausgabe der Register für π
// ====================================

public static String ausgabePiString(boolean bloecke) {
	String s = "";
	s = Long.toString(regGesSum[0]) + "," + "\n";
	String kette = "";

	for (int j = 1; j < anzahlRegs - cZusatzRegister; j++) {
		kette = String.valueOf(regGesSum[j]);
		while (kette.length() < cRegStellen)
			kette = '0' + kette;
		s += kette;
		if (bloecke)
			s += " "; // Lücke zur Trennung der Register !
	}
	// letztes Register bearbeiten
	kette = String.valueOf(regGesSum[anzahlRegs - cZusatzRegister]);
		while (kette.length() < cRegStellen)
			kette = '0' + kette;
	// System.out.println(kette);
	if (cRestStellen != 0)
	   kette = kette.substring(0, cRestStellen);

	s = s + kette;

	return s;
}

// ========================================
//              Berechnungen
// ========================================

// MACHIN-Reihe:   π = 4 · ( 4 · arctan(1/5) - arctan(1/239) )
// arctan(x) = x - x^3/3 + x^5/5 - x^7/7 + ... +(-1)^n·x^(2n+1)/(2n+1)

static int k5max,k239max,anzahlRegs,anzRegsplus1, gesSt;
static double log5,log239;
static long[] reg1, regSum, regGesSum, regXpot, regXdurchK;
static int cRestStellen;

public static void piBerechnung(int gesStellen) { // π-Gesamtterm berechnen
	// 1. Parameter initialisieren:
	init(gesStellen);
	regXdurchK = new long[anzRegsplus1];
	regGesSum = new long[anzRegsplus1];
	log5 = Math.log10(5); // 0,69
	log239 = Math.log10(239); // 2,37
	// 2·n5 + 1 ≥ [( gesSt + lg(16/2) ) / lg(5) - 1] / 2
	k5max = (int) ((gesSt + Math.log10(8)) / log5) + 1;
	// n239 ≥ [( gesSt + lg(4/2) ) / lg(239) - 1] / 2
	k239max = (int) ((gesSt + Math.log10(2)) / log239) + 1;

	// 2. eigentliche Berechnung (der Reihen) von π
	arctan1durch5();  // Ergebnis ist regSum
	regGesSum = RegSumme(regSum, regSum, anzahlRegs,0); // 2·atan/1/5)
	regGesSum = RegSumme(regGesSum, regGesSum, anzahlRegs,0); // 4·atan(1/5)
	arctan1durch239();  // Ergebnis ist wieder regSum
	regGesSum = RegDifferenz(regGesSum, regSum, anzahlRegs,0); // 4·atan(1/5)-atan(1/239)

	regGesSum = RegSumme(regGesSum, regGesSum, anzahlRegs,0); // 2·(4·atan(1/5)-atan(1/239))
	regGesSum = RegSumme(regGesSum, regGesSum, anzahlRegs,0); // 4·(4·atan(1/5)-atan(1/239))
}

public static void init(int gesStellen) {
	cRestStellen = gesStellen % cRegStellen;
	gesSt = gesStellen + cZusatzRegister*cRegStellen;
	anzahlRegs = gesSt / cRegStellen;
	if (cRestStellen > 0)
		anzahlRegs++;
	anzRegsplus1 = anzahlRegs + 1; // Register 0 mitzählen !!
}
```

```
	public static void arctan1durch5() {
		// berechnet  ∑ ( (-1)^k · (1/5)^(2k+1) / (2k+1), k , 0 , n )
		regXpot = new long[anzRegsplus1]; // altes Feld löschen
		regXpot[0] = 1L; // Zähler = 1 für Kehrwertbildung !
		regXpot = RegQuotient(regXpot, 5, anzahlRegs, 0); //  = 1/5
		regSum = Arrays.copyOf(regXpot, anzRegsplus1); // sum = 1/5
		boolean negativ = false;
		int nStart, zweikp1 = 1; // zweikp1 = 2k+1
		while (zweikp1 < k5max) {
			zweikp1 += 2;  // 3; 5; 7: ...
			nStart = (int) ((log5*zweikp1 + Math.log10(zweikp1))/cRegStellen);
			regXpot = RegQuotient(regXpot, 25, anzahlRegs, nStart); //  (1/5) / 5² / 5² ...
			regXdurchK = Arrays.copyOf(regXpot, anzRegsplus1);
			// Kopie notwendig, damit regXpot für den nächsten Umlauf gleich bleibt
			// RegXpot wird nur für die Potenzen (1/5)^(2k+1) verwendet !!
			regXdurchK = RegQuotient(regXdurchK, zweikp1, anzahlRegs, nStart);
			negativ = !negativ;
			if (negativ)
				regSum = RegDifferenz(regSum, regXdurchK, anzahlRegs, nStart);
			else // Vorzeichen +
				regSum = RegSumme(regSum, regXdurchK, anzahlRegs, nStart);
		} // while
	}

	public static void arctan1durch239() {
		// berechnet  ∑ ( (-1)^k · (1/239)^(2k+1) / (2k+1), k , 0 , n )
		regXpot = new long[anzRegsplus1]; // altes Feld löschen
		regXpot[0] = 1L; // Zahl 1 für Kehrwertbildung !
		regXpot = RegQuotient(regXpot, 239, anzahlRegs,0); //  = 1/239
		regSum = Arrays.copyOf(regXpot, anzRegsplus1);// sum = 1/239
		boolean negativ = false;
		int nStart, zweikp1 = 1; // zweikp1 = 2k+1
		while (zweikp1< k239max) {
			zweikp1 += 2;  // 3; 5; 7: ...
			nStart = (int) ((log239*zweikp1 + Math.log10(zweikp1))/cRegStellen);
			regXpot = RegQuotient(regXpot, 57121, anzahlRegs, nStart); // 57121 = 239²
			regXdurchK = Arrays.copyOf(regXpot, anzRegsplus1);
			regXdurchK = RegQuotient(regXdurchK, zweikp1, anzahlRegs, nStart);
			negativ = !negativ;
			if (negativ)
				regSum = RegDifferenz(regSum, regXdurchK, anzahlRegs, nStart);
			else // Vorzeichen +
				regSum = RegSumme(regSum, regXdurchK, anzahlRegs, nStart);
		} // while
	}
}
```