

# Generatoren programmieren (Fortsetzung)

Hier noch einmal der Generator der Primzahlzwillinge vom letzten Mal.

```
In [1]: def primzw(n=oo):
    p=3
    while p<n:
        q=next_prime(p)
        if q==p+2:
            yield (p,q)
        p=q
```

```
In [2]: list(primzw(1000))
```

```
Out[2]: [(3, 5),
          (5, 7),
          (11, 13),
          (17, 19),
          (29, 31),
          (41, 43),
          (59, 61),
          (71, 73),
          (101, 103),
          (107, 109),
          (137, 139),
          (149, 151),
          (179, 181),
          (191, 193),
          (197, 199),
          (227, 229),
          (239, 241),
          (269, 271),
          (281, 283),
          (311, 313),
          (347, 349),
          (419, 421),
          (431, 433),
          (461, 463),
          (521, 523),
          (569, 571),
          (599, 601),
          (617, 619),
          (641, 643),
          (659, 661),
          (809, 811),
          (821, 823),
          (827, 829),
          (857, 859),
          (881, 883)]
```

Man kann aus gegebenen Generatoren auch neue programmieren. Hier ist ein Generator, der die ersten n Elemente des gegebenen Generators g liefert.

```
In [3]: def firstn(g,n):
    for i in range(n):
        yield next(g)
```

Die ersten 20 Primzahlzwillinge.

```
In [4]: list(firstn(primzw(),20))
```

```
Out[4]: [(3, 5),
          (5, 7),
          (11, 13),
          (17, 19),
          (29, 31),
          (41, 43),
          (59, 61),
          (71, 73),
          (101, 103),
          (107, 109),
          (137, 139),
          (149, 151),
          (179, 181),
          (191, 193),
          (197, 199),
          (227, 229),
          (239, 241),
          (269, 271),
          (281, 283),
          (311, 313)]
```

## Rekursive Generatoren

Generatoren können auch rekursiv definiert werden.

Im folgenden Beispiel schreiben wir einen Generator, der die Elemente der  $S^n$ -ten kartesischen Potenz einer Menge  $S$  liefert. Diese kann induktiv definiert werden durch  $S^n = S \times S^{n-1}$ . Das lässt sich direkt in einen Generator übersetzen.

```
In [5]: def cart(S,n):
    if n==0:
        yield []
    else:
        for s in S:
            for p in cart(S,n-1):
                yield [s]+p
```

```
In [6]: list(cart([1,2],4))
```

```
Out[6]: [[1, 1, 1, 1],  
         [1, 1, 1, 2],  
         [1, 1, 2, 1],  
         [1, 1, 2, 2],  
         [1, 2, 1, 1],  
         [1, 2, 1, 2],  
         [1, 2, 2, 1],  
         [1, 2, 2, 2],  
         [2, 1, 1, 1],  
         [2, 1, 1, 2],  
         [2, 1, 2, 1],  
         [2, 1, 2, 2],  
         [2, 2, 1, 1],  
         [2, 2, 1, 2],  
         [2, 2, 2, 1],  
         [2, 2, 2, 2]]
```

Hier schreiben wir einen Generator, der alle Permutationen einer Menge  $S$  liefert (hier dargestellt als alle Anordnungen der Elemente von  $S$ ). Jede Permutation besteht aus einem Element, das an den Anfang gestellt wird, und einer Permutation der restlichen Elemente. Daraus ergibt sich folgender rekursiver Generator.

```
In [7]: def perm(S):  
    if S==Set(): # leere Menge  
        yield []  
    else:  
        for s in S:  
            for p in perm(S-{s}):  
                yield [s]+p
```

```
In [8]: list(perm({1,2,3,4}))
```

```
Out[8]: [[1, 2, 3, 4],  
         [1, 2, 4, 3],  
         [1, 3, 2, 4],  
         [1, 3, 4, 2],  
         [1, 4, 2, 3],  
         [1, 4, 3, 2],  
         [2, 1, 3, 4],  
         [2, 1, 4, 3],  
         [2, 3, 1, 4],  
         [2, 3, 4, 1],  
         [2, 4, 1, 3],  
         [2, 4, 3, 1],  
         [3, 1, 2, 4],  
         [3, 1, 4, 2],  
         [3, 2, 1, 4],  
         [3, 2, 4, 1],  
         [3, 4, 1, 2],  
         [3, 4, 2, 1],  
         [4, 1, 2, 3],  
         [4, 1, 3, 2],  
         [4, 2, 1, 3],  
         [4, 2, 3, 1],  
         [4, 3, 1, 2],  
         [4, 3, 2, 1]]
```

# Fehlerbehandlungen: Exceptions

Wir haben schon verschiedene Fehlermeldungen gesehen. Diese sind in Python eigene Objekte, sogenannte **Exceptions**. Exceptions können je nach Art des Fehlers verschiedenen Klassen angehören, z.B. NameError, ValueError etc.

In [9]:

```
y
```

```
-> NameError  
      t)  
Cell In [9], line 1  
----> 1 y
```

Traceback (most recent call last)

NameError: name 'y' is not defined

In [10]:

```
factorial(-1)
```

```
-          Traceback (most recent call last)
-
ValueError
t)
Cell In [10], line 1
----> 1 factorial(-Integer(1))

File /opt/sagemath/sage-10.1/src/sage/symbolic/function.pyx:1033, in sage.
symbolic.function.BuiltinFunction.__call__()
    1031 res = self._evalf_try_(*args)
    1032 if res is None:
-> 1033     res = super().__call__(
    1034             *args, coerce=coerce, hold=hold)
    1035

File /opt/sagemath/sage-10.1/src/sage/symbolic/function.pyx:547, in sage.s
ymbolic.function.Function.__call__()
    545         raise TypeError("arguments must be symbolic expression
s")
    546
-> 547 return call_registered_function(self._serial, self._nargs, args, h
old,
    548                     not symbolic_input, SR)
    549

File /opt/sagemath/sage-10.1/src/sage/symbolic/pynac_function_impl.pxi:1,
in sage.symbolic.expression.call_registered_function()
----> 1 cpdef call_registered_function(unsigned serial,
    2                     int nargs,
    3                     list args,

File /opt/sagemath/sage-10.1/src/sage/symbolic/pynac_function_impl.pxi:49,
in sage.symbolic.expression.call_registered_function()
    47     res = g_function_evalv(serial, vec, hold)
    48 elif nargs == 1:
---> 49     res = g_function_eval1(serial,
    50             (<Expression>args[0])._gobj, hold)
    51 elif nargs == 2:

File /opt/sagemath/sage-10.1/src/sage/functions/other.py:1523, in Function
_factorial._eval_(self, x)
    1521 if isinstance(x, (int, Integer)):
    1522     try:
-> 1523         return x.factorial()
    1524     except OverflowError:
    1525         return

File /opt/sagemath/sage-10.1/src/sage/rings/integer.pyx:4516, in sage.ring
s.integer.Integer.factorial()
    4514 """
    4515 if mpz_sgn(self.value) < 0:
-> 4516     raise ValueError("factorial only defined for non-negative inte
gers")
    4517
    4518 if not mpz.fits_ulong_p(self.value):
```

ValueError: factorial only defined for non-negative integers

In [11]: 1/0

```

-----
-
ZeroDivisionError                                Traceback (most recent call last)
t)
Cell In [11], line 1
----> 1 Integer(1)/Integer(0)

File /opt/sagemath/sage-10.1/src/sage/rings/integer.pyx:2015, in sage.ring
s.integer.Integer.__truediv__()
  2013 if type(left) is type(right):
  2014     if mpz_sgn((<Integer>right).value) == 0:
-> 2015         raise ZeroDivisionError("rational division by zero")
  2016     x = <Rational> Rational.__new__(Rational)
  2017     mpq_div_zz(x.value, (<Integer>left).value, (<Integer>right).va
lue)

ZeroDivisionError: rational division by zero

In [12]: 1/[1,2]
-----
-
TypeError                                Traceback (most recent call last)
t)
Cell In [12], line 1
----> 1 Integer(1)/[Integer(1),Integer(2)]

File /opt/sagemath/sage-10.1/src/sage/rings/integer.pyx:2030, in sage.ring
s.integer.Integer.__truediv__()
  2028     return y
  2029
-> 2030     return coercion_model.bin_op(left, right, operator.truediv)
  2031
  2032 cpdef __div__(self, right):

File /opt/sagemath/sage-10.1/src/sage/structure/coerce.pyx:1269, in sage.s
tructure.coerce.CoercionModel.bin_op()
  1267     # We should really include the underlying error.
  1268     # This causes so much headache.
-> 1269     raise bin_op_exception(op, x, y)
  1270
  1271 cpdef canonical_coercion(self, x, y):


TypeError: unsupported operand parent(s) for /: 'Integer Ring' and '<class
'list'>'
```

## Fehlermeldungen ausgeben

Wir sehen nun, wie man in Python professionell Fehlermeldungen ausgibt.

```
In [13]: def fac(n):
    if n==0:
        return 1
    return n*fac(n-1)
```

```
In [14]: fac(6)
```

```
Out[14]: 720
```

```
In [15]: fac(5)
```

```
Out[15]: 120
```

```
In [16]: fac(-1)
```

```
-  
RecursionError  
t)  
Cell In [16], line 1  
----> 1 fac(-Integer(1))  
  
Cell In [13], line 4, in fac(n)  
    2 if n==Integer(0):  
    3     return Integer(1)  
----> 4 return n*fac(n-Integer(1))  
  
Cell In [13], line 4, in fac(n)  
    2 if n==Integer(0):  
    3     return Integer(1)  
----> 4 return n*fac(n-Integer(1))  
  
[... skipping similar frames: fac at line 4 (2969 times)]  
  
Cell In [13], line 4, in fac(n)  
    2 if n==Integer(0):  
    3     return Integer(1)  
----> 4 return n*fac(n-Integer(1))  
  
Cell In [13], line 2, in fac(n)  
    1 def fac(n):  
----> 2     if n==Integer(0):  
    3         return Integer(1)  
    4     return n*fac(n-Integer(1))
```

RecursionError: maximum recursion depth exceeded while calling a Python object

Hier hat `fac(-1)` zu einer Endlosrekursion geführt. Um das zu vermeiden, müssen wir abfragen, ob das Argument negativ ist. Hier zuerst die unprofessionelle Variante, eine Fehlermeldung als String zurückzugeben.

```
In [17]: def fac(n): # unprofessionell  
    if n<0:  
        return "n ist negativ"  
    if n==0:  
        return 1  
    return n*fac(n-1)
```

```
In [18]: fac(-1)
```

```
Out[18]: 'n ist negativ'
```

Nachteil: im weiteren Programmverlauf kann es zu Fehlern oder seltsamem Verhalten kommen, wenn der zurückgegebene Wert statt der erwarteten Zahl ein String ist.

```
In [19]: m=fac(-1)
```

```
In [20]: m^2
```

```
-  
TypeError  
t)  
Cell In [20], line 1  
----> 1 m**Integer(2)  
  
File /opt/sagemath/sage-10.1/src/sage/rings/integer.pyx:2188, in sage.ring.  
s.integer.Integer.__pow__()  
    2186         return coercion_model.bin_op(left, right, operator.pow)  
    2187     # left is a non-Element: do the powering with a Python int  
-> 2188     return left ** int(right)  
    2189  
    2190 cpdef __pow__(self, other):  
  
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

```
In [21]: m+m
```

```
Out[21]: 'n ist negativ ist negativ'
```

Hier die professionelle Variante: mit dem Schlüsselwort **raise** wird eine Exception der Klasse ValueError erstellt.

```
In [22]: def fac(n):  
    if n<0:  
        raise ValueError('n ist negativ')  
    if n==0:  
        return 1  
    return n*fac(n-1)
```

Das Ergebnis sieht nun aus wie die Fehlermeldungen der internen Python- oder Sage-Funktionen, die wir weiter oben gesehen haben.

```
In [23]: fac(-1)
```

```
-  
  ValueError  
t)  
Cell In [23], line 1  
----> 1 fac(-Integer(1))  
  
Cell In [22], line 3, in fac(n)  
  1 def fac(n):  
  2     if n<Integer(0):  
----> 3         raise ValueError('n ist negativ')  
  4     if n==Integer(0):  
  5         return Integer(1)  
  
ValueError: n ist negativ
```

Der Vorteil von Exceptions ist, dass diese im umgebenden Code gezielt abgefangen werden können. Mit **try** wird versucht, einen Codeblock auszuführen. Darauf folgende **except**-Blöcke werden dann ausgeführt, wenn im **try**-Block eine entsprechende Exception auftritt. Damit können Fehler gezielt behandelt werden.

Hier als Beispiel eine Funktion, die bei negativen Zahlen  $n$  die Faktorielle von  $-n$  zurückgeben soll.

```
In [24]: def fac1(n):  
    try:  
        return fac(n)  
    except ValueError:  
        print('failed')  
        return fac(-n)
```

```
In [25]: fac1(6)
```

```
Out[25]: 720
```

```
In [26]: fac1(-3)
```

```
failed
```

```
Out[26]: 6
```

In der Definition von  $\text{fac1}(n)$  gibt es bisher nur einen **except**-Block für `ValueError`. Andere Exceptions werden nicht abgefangen und kommen daher durch bis zum Benutzer.

```
In [27]: fac1([1,2])
```

```

-----
-
TypeError                                         Traceback (most recent call las
t)
Cell In [27], line 1
----> 1 fac1([Integer(1),Integer(2)])

Cell In [24], line 3, in fac1(n)
    1 def fac1(n):
    2     try:
----> 3         return fac(n)
    4     except ValueError:
    5         print('failed')

Cell In [22], line 2, in fac(n)
    1 def fac(n):
----> 2     if n<Integer(0):
    3         raise ValueError('n ist negativ')
    4     if n==Integer(0):

File /opt/sagemath/sage-10.1/src/sage/rings/integer.pyx:912, in sage.ring
s.integer.Integer.__richcmp__()
    910     c = mpz_cmp_d((<Integer>left).value, d)
    911 else:
--> 912     return coercion_model.richcmp(left, right, op)
    913
    914 return rich_to_bool_sgn(op, c)

File /opt/sagemath/sage-10.1/src/sage/structure/coerce.pyx:2038, in sage.s
tructure.coerce.CoercionModel.richcmp()
    2036     raise bin_op_exception('<=' , x, y)
    2037 elif op == Py_GT:
-> 2038     raise bin_op_exception('>' , x, y)
    2039 else:
    2040     raise bin_op_exception('>=' , x, y)

TypeError: unsupported operand parent(s) for >: 'Integer Ring' and '<class
'list'>'
```

Man kann auch mehrere Exceptions abfangen. Mit **as** kann man die abgefangene Exception einer Variable zuweisen, um im darauffolgenden Block auf sie zuzugreifen. **except:** ohne Angabe einer Klasse fängt alle Exceptions ab.

```
In [28]: def fac1(n):
    try:
        return fac(n)
    except ValueError:
        print('failed')
        return fac(-n)
    except TypeError as e:
        print('sinnloses Argument')
        print(e.args[0])
    except:
        print('etwas ging schief')
```

```
In [29]: fac1([1,2])
```

```
sinnloses Argument
unsupported operand parent(s) for >: 'Integer Ring' and '<class 'list'>'
```

```
In [30]: fac1(oo)
```

```
etwas ging schief
```

## Klassen programmieren

Wir sehen und kurz an, wie man in Python eigene Klassen programmieren kann.

Eine Klasse stellt Funktionen (sogenannte Methoden) bereit, die man mit ihren Objekten ausführen kann.

Wir wollen als Beispiel eine Klasse MaxPlus definieren, die die Max-Plus-Algebra  $(\mathbb{R} \cup -\infty, \oplus, \odot)$  implementiert, also den Halbring mit Operationen  $a \oplus b = \max\{a, b\}$  und  $a \odot b = a + b$ . Die Operationen sollen wie gewohnt direkt mit `+` und `*` aufgerufen werden.

Intern wird bei  $a+b$  die Methode `__add__(b)` der jeweiligen Klasse aufgerufen.

```
In [31]: 1+3
```

```
Out[31]: 4
```

```
In [32]: 1.__add__(3)
```

```
Out[32]: 4
```

Jede Klasse soll zumindest zwei Methoden haben: `__init__(v)`, die bei der Erstellung neuer Objekte der Klasse aufgerufen wird, und `__repr__(self)`, die angibt, wie Objekte im Output dargestellt werden.

Hier erhält das Objekt bei der Erstellung ein Attribut `.val`, in dem der übergebene Wert `v` (= die Zahl, die das Objekt darstellen soll) gespeichert wird.

Zur Darstellung wird der Wert in eckigen Klammern ausgegeben, um Elemente der Max-Plus-Algebra von herkömmlichen Zahlen zu unterscheiden.

```
In [33]: class MaxPlus:  
    def __init__(self, v):  
        self.val=v  
    def __repr__(self):  
        return f"[{self.val}]"
```

```
In [34]: a=MaxPlus(-1)
```

```
In [35]: a
```

```
Out[35]: [-1]
```

```
In [36]: a.val
```

```
Out[36]: -1
```

```
In [37]: b=MaxPlus(-3)
```

```
In [38]: b
```

```
Out[38]: [-3]
```

```
In [39]: a+b
```

```
-
```

```
TypeError
```

```
t)
```

```
Cell In [39], line 1
```

```
----> 1 a+b
```

```
Traceback (most recent call last)
```

```
TypeError: unsupported operand type(s) for +: 'MaxPlus' and 'MaxPlus'
```

Die Operationen Addition und Multiplikation müssen wir erst implementieren.

```
In [40]: class MaxPlus:  
    def __init__(self,v):  
        self.val=v  
    def __repr__(self):  
        return f"[{self.val}]"  
    def __add__(self,b):  
        return MaxPlus(max(self.val,b.val))  
    def __mul__(self,b):  
        return MaxPlus(self.val+b.val)
```

```
In [41]: a=MaxPlus(-1)  
b=MaxPlus(-3)
```

```
In [42]: a+b
```

```
Out[42]: [-1]
```

```
In [43]: a*b
```

```
Out[43]: [-4]
```

Tatsächlich ist Max-Plus-Algebra in Sage bereits implementiert!

```
In [ ]: TropicalSemiring?
```

```
In [44]: MP=TropicalSemiring(QQ,use_min=False)  
MP
```

```
Out[44]: Tropical semiring over Rational Field
```

```
In [45]: a=MP(-1)  
b=MP(-3)
```

```
In [46]: a+b
```

```
Out[46]: -1
```

```
In [47]: a*b
```

```
Out[47]: -4
```

Achtung: nicht alle Sage-Funktionen verhalten sich wie erwartet:

```
In [48]: add([a,b])
```

```
Out[48]: 0
```

```
In [49]: add?
```

**Signature:** add(iterable, /, start=0)

**Docstring:**

Return the sum of a 'start' value (default: 0) plus an iterable of numbers

When the iterable is empty, return the start value. This function is intended specifically for use with numeric values and may reject non-numeric types.

**Init docstring:** Initialize self. See help(type(self)) for accurate signature.

**File:**

**Type:** builtin\_function\_or\_method

```
In [50]: 0+a
```

```
Out[50]: 0
```

Das Problem ist, dass **add** die Summe standardmäßig mit \$0\$ beginnt, aber \$0\$ nicht das neutrale Element in MP ist.

```
In [51]: MP.zero()
```

```
Out[51]: -infinity
```

```
In [52]: add([a,b],start=MP.zero())
```

```
Out[52]: -1
```

## Mathematische Strukturen in Sage

Sehr viele mathematische Strukturen sind bereits in Sage implementiert. Wir sehen uns hier exemplarisch zwei Strukturen an: Gruppen, am Beispiel der symmetrischen Gruppe  $S_n$  und Graphen.

## Symmetrische Gruppe

```
In [53]: S5=SymmetricGroup(5)
```

```
In [54]: S5
```

```
Out[54]: Symmetric group of order 5! as a permutation group
```

Angabe von Permutationen  $\sigma \in S_n$  in Listenschreibweise:  $[\sigma(1), \sigma(2), \dots, \sigma(n)]$

```
In [55]: p1=S5([2,1,4,5,3])  
p1
```

```
Out[55]: (1,2)(3,4,5)
```

Intern werden Permutationen in **Zykenschreibweise** umgewandelt. Ein **Zyklus**  $\sigma = (a_1, a_2, \dots, a_k)$  ist die Permutation mit  $\sigma(a_1) = a_2, \sigma(a_2) = a_3, \dots, \sigma(a_{k-1}) = a_k$  und  $\sigma(a_k) = a_1$ . Jede Permutation kann als Produkt disjunkter Zyklen geschrieben werden.

```
In [56]: S5([2,4,5,6,1,3])
```

```
- Assertion Traceback (most recent call last)
t)
Cell In [56], line 1
----> 1 S5([Integer(2), Integer(4), Integer(5), Integer(6), Integer(1), Integer(3)])
      
```

File /opt/sagemath/sage-10.1/src/sage/structure/parent.pyx:901, in sage.structure.parent.Parent.\_\_call\_\_()  
899 if mor is not None:  
900 if no\_extra\_args:  
--> 901 return mor.\_\_call\_\_(x)  
902 else:  
903 return mor.\_\_call\_with\_args(x, args, kwds)

File /opt/sagemath/sage-10.1/src/sage/structure/coerce\_maps.pyx:163, in sage.structure.coerce\_maps.DefaultConvertMap\_unique.\_\_call\_\_()  
161 print(type(C), C)  
162 print(type(C.\_element\_constructor), C.\_element\_constructor)  
163 raise  
164  
165 cpdef Element \_\_call\_with\_args(self, x, args=(), kwds={}):

File /opt/sagemath/sage-10.1/src/sage/structure/coerce\_maps.pyx:158, in sage.structure.coerce\_maps.DefaultConvertMap\_unique.\_\_call\_\_()  
156 cdef Parent C = self.\_codomain  
157 try:  
--> 158 return C.\_element\_constructor(x)  
159 except Exception:  
160 if print\_warnings:

File /opt/sagemath/sage-10.1/src/sage/groups/perm\_gps/permgroup.py:884, in PermutationGroup\_generic.\_element\_constructor\_(self, x, check)  
880 if compatible\_domains and (isinstance(self, SymmetricGroup)  
881 or x.gap() in self.gap()):  
882 return self.element\_class(x, self, check=False)  
--> 884 return self.element\_class(x, self, check=check)

File /opt/sagemath/sage-10.1/src/sage/groups/perm\_gps/permgroup\_element.pyx:470, in sage.groups.perm\_gps.permgroup\_element.PermutationGroupElement.\_\_init\_\_()  
468 self.\_set\_list\_cycles(g, convert)  
469 else:  
--> 470 self.\_set\_list\_images(g, convert)  
471 elif isinstance(g, str):  
472 self.\_set\_string(g)

File /opt/sagemath/sage-10.1/src/sage/groups/perm\_gps/permgroup\_element.pyx:558, in sage.groups.perm\_gps.permgroup\_element.PermutationGroupElement.\_set\_list\_images()  
556 """
557 cdef int i, j, vn = len(v)
--> 558 assert(vn <= self.n)
559 if convert:
560 convert\_dict = self.\_parent.\_domain\_to\_gap

AssertionError:

```
In [57]: S5([1,1,2,3,4])
```

```
-----
-
ValueError                                Traceback (most recent call last)
t)
Cell In [57], line 1
----> 1 S5([Integer(1),Integer(1),Integer(2),Integer(3),Integer(4)])

File /opt/sagemath/sage-10.1/src/sage/structure/parent.pyx:901, in sage.structure.parent.Parent.__call__()
    899 if mor is not None:
    900     if no_extra_args:
--> 901         return mor._call_(x)
    902     else:
    903         return mor._call_with_args(x, args, kwds)

File /opt/sagemath/sage-10.1/src/sage/structure/coerce_maps.pyx:163, in sage.structure.coerce_maps.DefaultConvertMap_unique.__call__()
    161         print(type(C), C)
    162         print(type(C._element_constructor), C._element_constructor)
cstor)
--> 163         raise
    164
    165 cpdef Element __call__(self, x, args=(), kwds={}):
    166
    167     if print_warnings:
    168         print("Warning: coercing from", type(x))

File /opt/sagemath/sage-10.1/src/sage/structure/coerce_maps.pyx:158, in sage.structure.coerce_maps.DefaultConvertMap_unique.__call__()
    156 cdef Parent C = self._codomain
    157 try:
--> 158     return C._element_constructor(x)
    159 except Exception:
    160     if print_warnings:
    161         print("Warning: coercing from", type(x))

File /opt/sagemath/sage-10.1/src/sage/groups/perm_gps/permgroup.py:884, in PermutationGroup_generic._element_constructor_(self, x, check)
    880     if compatible_domains and (isinstance(self, SymmetricGroup)
    881                                 or x.gap() in self.gap()):
    882         return self.element_class(x, self, check=False)
--> 884 return self.element_class(x, self, check=check)

File /opt/sagemath/sage-10.1/src/sage/groups/perm_gps/permgroup_element.py:508, in sage.groups.perm_gps.permgroup_element.PermutationGroupElement.__init__()
    506 # a valid permutation (else segfaults, infinite loops may occur).
    507 if not is_valid_permutation(self.perm, self.n):
--> 508     raise ValueError("invalid data to initialize a permutation")
    509
    510 # This is more expensive

ValueError: invalid data to initialize a permutation
```

Auch die Eingabe kann in Zyklenschreibweise, als Liste der Zyklen, erfolgen.

```
In [58]: p2=S5([(2,3),(1,4)])
p2
```

```
Out[58]: (1,4)(2,3)
```

```
In [59]: p1*p2
```

```
Out[59]: (1,3)(2,4,5)
```

```
In [60]: p2*p1
```

```
Out[60]: (1,5,3)(2,4)
```

```
In [61]: p1^-1
```

```
Out[61]: (1,2)(3,5,4)
```

```
In [62]: p1
```

```
Out[62]: (1,2)(3,4,5)
```

Analog zu Vektorräumen: die von Elementen erzeugte Untergruppe ist die kleinste Untergruppe, die alle gegebenen Elemente enthält.

```
In [63]: G=S5.subgroup([p1])
```

```
G
```

```
Out[63]: Subgroup generated by [(1,2)(3,4,5)] of (Symmetric group of order 5! as a permutation group)
```

```
In [64]: list(G)
```

```
Out[64]: [(), (1,2)(3,4,5), (3,5,4), (1,2), (3,4,5), (1,2)(3,5,4)]
```

```
In [65]: len(G)
```

```
Out[65]: 6
```

Verknüpfungstabelle

```
In [66]: G.cayley_table()
```

```
Out[66]: * a b c d e f  
+-----  
a| a b c d e f  
b| b c a e f d  
c| c a b f d e  
d| d e f a b c  
e| e f d b c a  
f| f d e c a b
```

```
In [67]: G.is_abelian()
```

```
Out[67]: True
```

```
In [68]: S5.is_abelian()
```

```
Out[68]: False
```

Liste aller Untergruppen

```
In [69]: S5ss=S5.subgroups()  
len(S5ss)
```

```
Out[69]: 156
```

```
In [70]: S5ss
```



Subgroup generated by  $[(1,3,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,4)(3,5), (2,5)(3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4)(2,5), (1,5)(2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3)(2,4), (1,4)(2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2)(3,5), (1,3)(2,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3)(4,5), (1,4)(3,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,5,3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,5,4,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3,5,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,2,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,3,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,3,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,4,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,4,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,5,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,5,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,4,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,5,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(4,5), (2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(4,5), (1,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(4,5), (1,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,4), (2,5)]$  of (Symmetric group of order 5! as a permutation group),

Subgroup generated by  $[(3,4), (1,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,4), (1,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,5), (2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,5), (1,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,5), (1,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3), (1,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3), (1,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,4), (1,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,4), (1,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,5), (1,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,5), (1,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,5,2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,5,3,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,4,5,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,5,2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,2,3,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,5,3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,4), (3,4,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,4), (2,4,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(4,5), (1,4,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3), (2,3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3), (1,3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,5), (2,3,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,5), (1,3,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2), (1,2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,4), (1,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,5), (1,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,4,5), (1,2)(3,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,3), (1,3)(4,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,5), (1,4)(2,3)]$  of (Symmetric group of order 5! as a permutation group),

Subgroup generated by  $[(2,3,4), (1,5)(2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3,5), (1,4)(2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,4,5), (1,3)(2,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,4)(3,5), (1,2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,5), (1,3)(2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,5)(3,4), (1,2,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,4), (1,4)(2,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,2)(4,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,2)(3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2)(3,5,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,2)(3,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,4)(2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5)(2,4,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4)(2,5,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,3)(2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3)(2,5,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,3)(2,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(4,5), (2,5)(3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2), (1,5)(2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(4,5), (1,5)(3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3)(4,5), (2,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,4), (1,4)(2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3)(4,5), (1,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,5), (2,5)(3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2), (1,5)(2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3)(4,5), (1,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3), (1,2)(3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3), (1,2)(3,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2)(4,5), (1,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,4), (1,4)(2,3)]$  of (Symmetric group of order 5! as a permutation group),

Subgroup generated by  $[(2,5), (1,2)(4,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2)(3,5), (1,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3)(2,5), (1,4,3,2,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,5,4,3), (1,3)(2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,4)(3,5), (1,5,4,2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,5,3,2), (1,4)(2,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4)(3,5), (1,5,3,4,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4)(2,3), (1,5,4,3,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3,4), (2,5)(3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,4,5), (1,3)(4,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,3), (1,4)(2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2)(3,5), (1,2,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,5), (1,5)(2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(4,5), (3,5,4), (1,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(4,5), (2,3), (1,3,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3), (1,5,4), (1,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,5,4), (2,5), (1,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,4), (2,4,3), (1,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,4), (1,2), (1,5,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,4), (2,5), (1,4,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3), (2,5,3), (1,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,5), (2,4), (1,4,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,4), (1,3), (1,5,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2)(3,4), (1,5,3,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3)(4,5), (1,4,3,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,5,4,3), (1,5)(3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,4,2), (1,5)(2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2)(4,5), (1,5,4,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,4,3), (1,3)(4,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3,5,4), (2,4,3)]$  of (Symmetric group of order 5! as a permutation group),

```
Subgroup generated by [(3,5,4), (1,5,3,4)] of (Symmetric group of order  
5! as a permutation group),  
Subgroup generated by [(1,4,5,2), (1,5,4)] of (Symmetric group of order  
5! as a permutation group),  
Subgroup generated by [(1,2,4,3), (1,3,2)] of (Symmetric group of order  
5! as a permutation group),  
Subgroup generated by [(1,3,2,5), (1,5,2)] of (Symmetric group of order  
5! as a permutation group),  
Subgroup generated by [(2,5,4), (1,4)(2,3)] of (Symmetric group of order  
5! as a permutation group),  
Subgroup generated by [(4,5), (1,2,4,3)] of (Symmetric group of order  
5! as a permutation group)]
```

Wir filtern diese Liste nach Abelschen Untergruppen.

```
In [71]: S5ab=[G for G in S5ss if G.is_abelian()]
```

```
In [72]: len(S5ab)
```

```
Out[72]: 87
```

```
In [73]: S5ab
```



Subgroup generated by  $[(1,3,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,4)(3,5), (2,5)(3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4)(2,5), (1,5)(2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3)(2,4), (1,4)(2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2)(3,5), (1,3)(2,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3)(4,5), (1,4)(3,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,5,3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,5,4,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3,5,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,2,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,3,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,3,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,4,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,4,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,5,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,5,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,4,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,5,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(4,5), (2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(4,5), (1,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(4,5), (1,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,4), (2,5)]$  of (Symmetric group of order 5! as a permutation group),

Subgroup generated by  $[(3,4), (1,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,4), (1,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,5), (2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,5), (1,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(3,5), (1,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3), (1,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,3), (1,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,4), (1,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,4), (1,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,5), (1,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(2,5), (1,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,5,2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,5,3,2)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,4,5,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,5,2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,2,3,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2,5,3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3,2)(4,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,2)(3,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,2)(3,5,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,2)(3,5)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,4)(2,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5)(2,4,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4)(2,5,3)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,5,3)(2,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,3)(2,5,4)]$  of (Symmetric group of order 5! as a permutation group),  
Subgroup generated by  $[(1,4,3)(2,5)]$  of (Symmetric group of order 5! as a permutation group)]

## Graphen

Ein Graph besteht aus einer Menge von Knoten und einer Menge von Kanten, die

zwischen je zwei Knoten verlaufen.

```
In [74]: g=Graph() # leerer Graph
```

```
In [75]: g.add_vertex(0)  
g
```

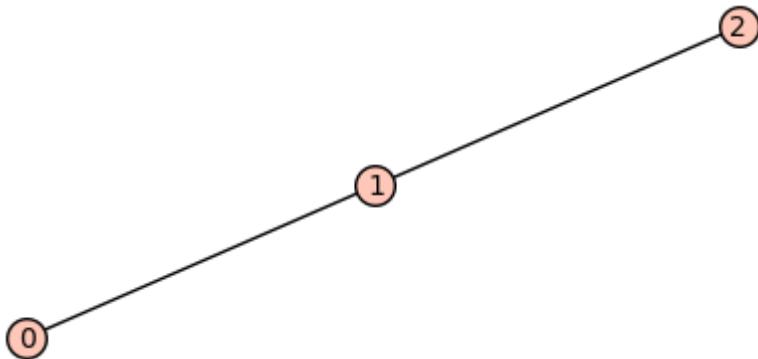
```
Out[75]: Graph on 1 vertex
```

A single red circular vertex labeled '0'.

```
In [76]: g.show()
```

0

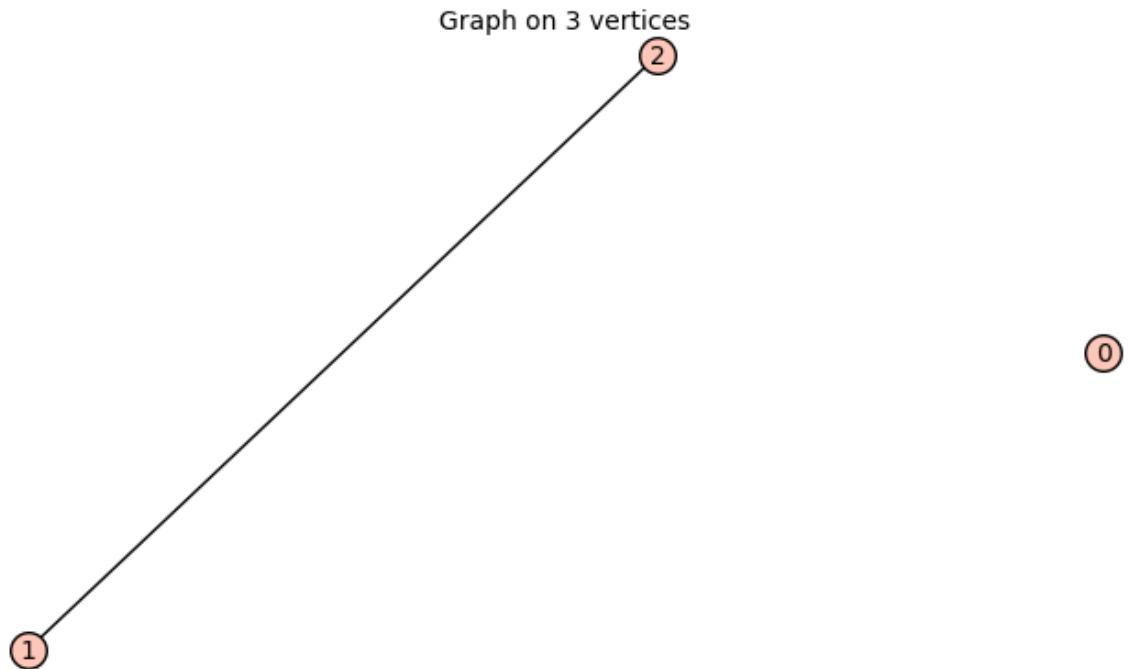
```
In [77]: g.add_edge(1,2)  
g.add_edge(1,0)  
g.show()
```



```
In [78]: g.delete_edge(1,0)
```

```
In [79]: g
```

Out[79]:



Als Bezeichnungen der Knoten können fast beliebige Objekte genommen werden.

Wichtig ist nur, dass die Objekte **immutable**, also unveränderbar sind.

Matrizen sind z.B. standardmäßig nicht immutable, da ihre Einträge verändert werden können.

```
In [80]: a=matrix(2,2,[1,x,2,3])  
a
```

```
Out[80]: [1 x]  
[2 3]
```

```
In [81]: g.add_edge(1,a)
```

```
-  
TypeError  
t)  
Cell In [81], line 1  
----> 1 g.add_edge(Integer(1),a)  
  
File /opt/sagemath/sage-10.1/src/sage/graphs/generic_graph.pyx:11676, in Ge  
nericGraph.add_edge(self, u, v, label)  
    11673         except Exception:  
    11674             pass  
> 11676 self._backend.add_edge(u, v, label, self._directed)  
  
File /opt/sagemath/sage-10.1/src/sage/graphs/base/c_graph.pyx:2360, in sag  
e.graphs.base.c_graph.CGraphBackend.add_edge()  
    2358         self.add_edge(u, v, l, directed)  
    2359  
> 2360 cpdef add_edge(self, object u, object v, object l, bint directed):  
    2361     """  
    2362     Add the edge ``(u,v)`` to self.  
  
File /opt/sagemath/sage-10.1/src/sage/graphs/base/c_graph.pyx:2438, in sag  
e.graphs.base.c_graph.CGraphBackend.add_edge()  
    2436  
    2437         cdef int u_int = self.check_labelled_vertex(u, False)  
> 2438         cdef int v_int = self.check_labelled_vertex(v, False)  
    2439  
    2440         cdef CGraph cg = self.cg()  
  
File /opt/sagemath/sage-10.1/src/sage/graphs/base/c_graph.pyx:1656, in sag  
e.graphs.base.c_graph.CGraphBackend.check_labelled_vertex()  
    1654 cdef CGraph G = self.cg()  
    1655  
> 1656 cdef int u_int = self.get_vertex(u)  
    1657 if u_int != -1:  
    1658     if not bitset_in(G.active_vertices, u_int):  
  
File /opt/sagemath/sage-10.1/src/sage/graphs/base/c_graph.pyx:1613, in sag  
e.graphs.base.c_graph.CGraphBackend.get_vertex()  
    1611 cdef CGraph G = self.cg()  
    1612 cdef long u_long  
> 1613 if u in vertex_ints:  
    1614     return vertex_ints[u]  
    1615 try:  
  
File /opt/sagemath/sage-10.1/src/sage/matrix/matrix0.pyx:5946, in sage.mat  
rix.matrix0.Matrix.__hash__()  
    5944 """  
    5945 if not self._is_immutable:  
> 5946     raise TypeError("mutable matrices are unhashable")  
    5947 if self.hash != -1:  
    5948     return self.hash
```

TypeError: mutable matrices are unhashable

Wir machen die Matrix immutable.

In [ ]: a.set\_immutable()

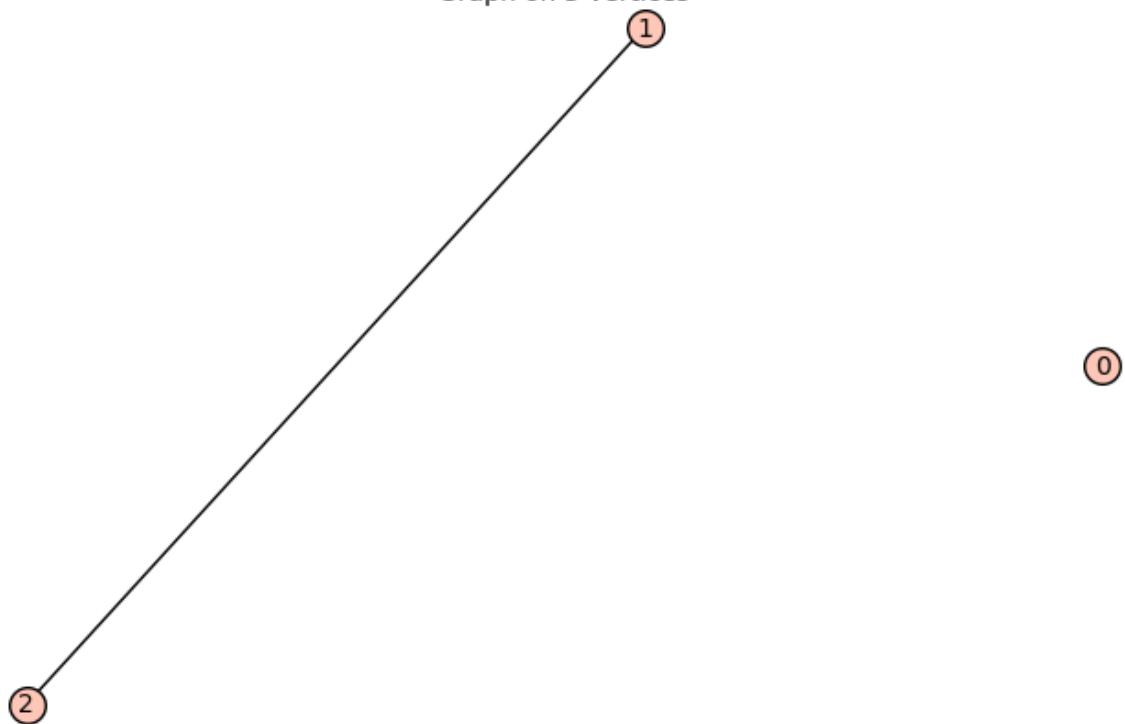
```
In [ ]: a[0,0]=2
```

```
In [ ]: g.add_edge(1,a)
```

```
In [82]: g
```

```
Out[82]:
```

Graph on 3 vertices

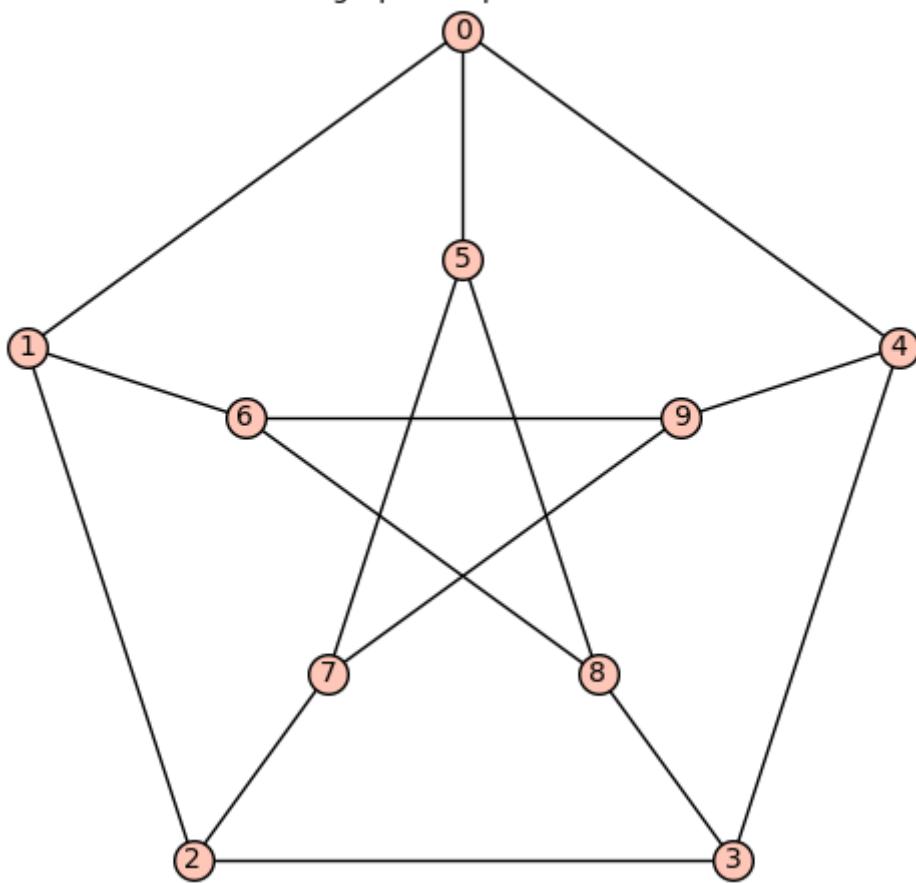


Sage hat eine große Bibliothek an vordefinierten Graphen.

```
In [83]: p=graphs.PetersenGraph()
p
```

```
Out[83]:
```

Petersen graph: Graph on 10 vertices



Viele Graphentheoretische Algorithmen sind in Sage implementiert. Hier ein paar Beispiele.

Kann der Graph gezeichnet werden, ohne dass sich Kanten überkreuzen?

```
In [84]: p.is_planar()
```

```
Out[84]: False
```

```
In [85]: p.neighbors(1)
```

```
Out[85]: [0, 2, 6]
```

Wie viele Farben braucht man, um die Knoten so einzufärben, dass benachbarte Knoten verschiedene Farben haben? (vgl. Vierfarbensatz)

```
In [86]: p.chromatic_number()
```

```
Out[86]: 3
```

Eine konkrete Färbung.

```
In [87]: p.coloring()
```

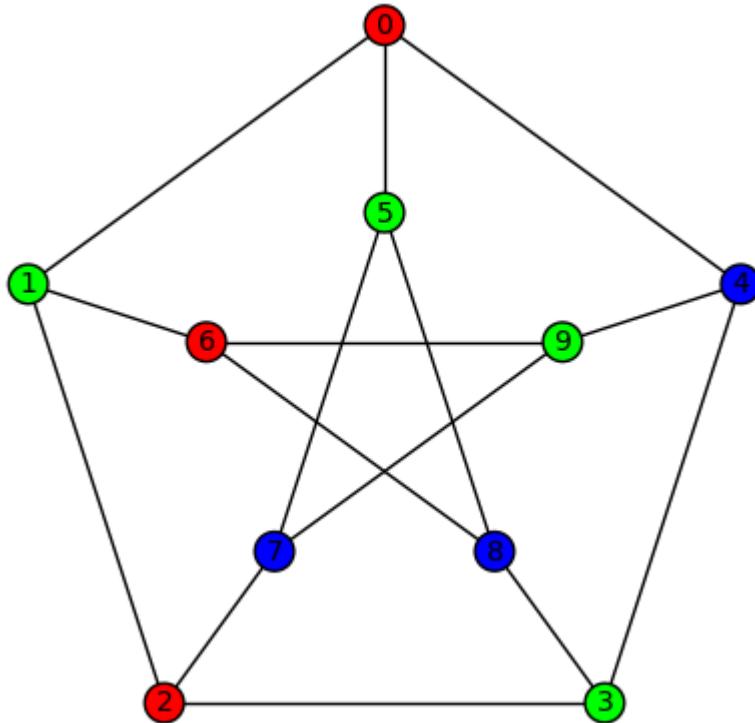
```
Out[87]: [[0, 2, 6], [1, 3, 5, 9], [4, 7, 8]]
```

Wir wollen die Färbung direkt im Bild des Graphen anzeigen.

```
In [88]: c=p.coloring(hex_colors=True)  
c
```

```
Out[88]: {'#ff0000': [0, 2, 6], '#00ff00': [1, 3, 5, 9], '#0000ff': [4, 7, 8]}
```

```
In [89]: p.show(vertex_colors=c)
```



Wir wollen nun die Knoten so umbenennen, dass jeweils die inneren mit den äußeren Knoten Plätze tauschen. Dass können wir über eine Permutation der Knoten erreichen.

```
In [90]: S10=SymmetricGroup(10)
```

```
In [91]: u=S10([6,7,8,9,10,1,2,3,4,5])  
u
```

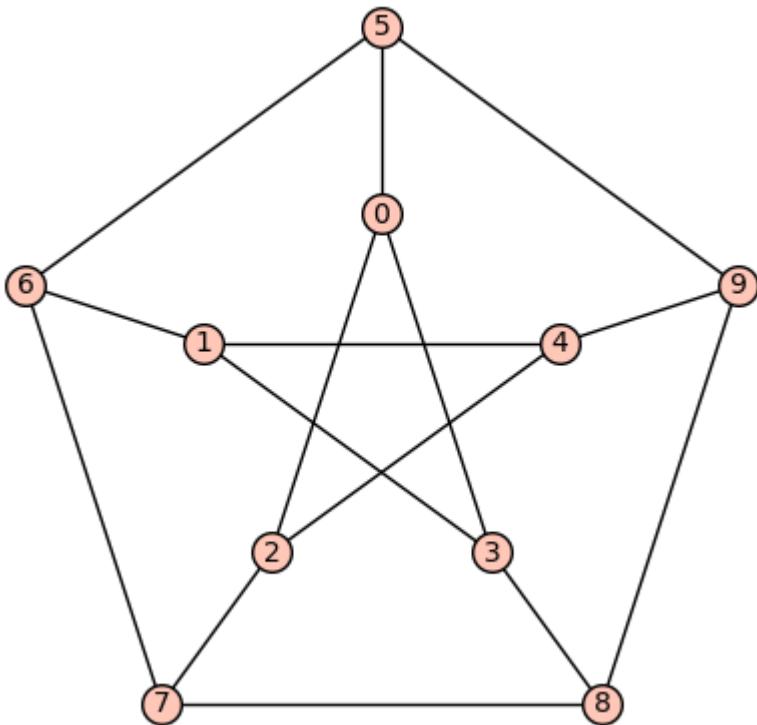
```
Out[91]: (1,6)(2,7)(3,8)(4,9)(5,10)
```

Umbenennen verändert den Graphen, also arbeiten wir lieber mit einer Kopie

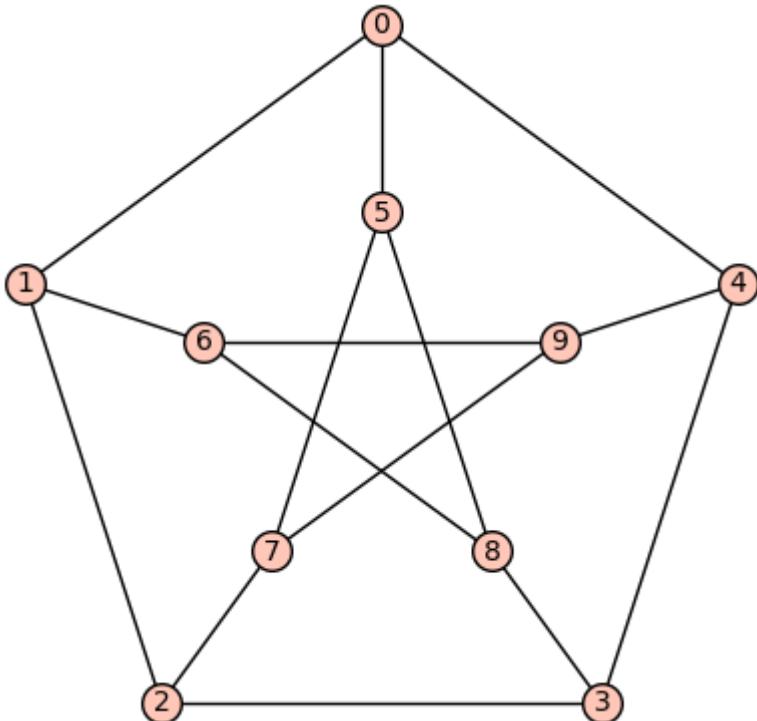
```
In [92]: p2=p.copy()
```

```
In [93]: p2.relabel(u)
```

```
In [94]: p2.show()
```



```
In [95]: p.show()
```



Sind zwei Graphen gleich bis auf Umbenennen der Knoten?

```
In [96]: p.is_isomorphic(p2)
```

```
Out[96]: True
```

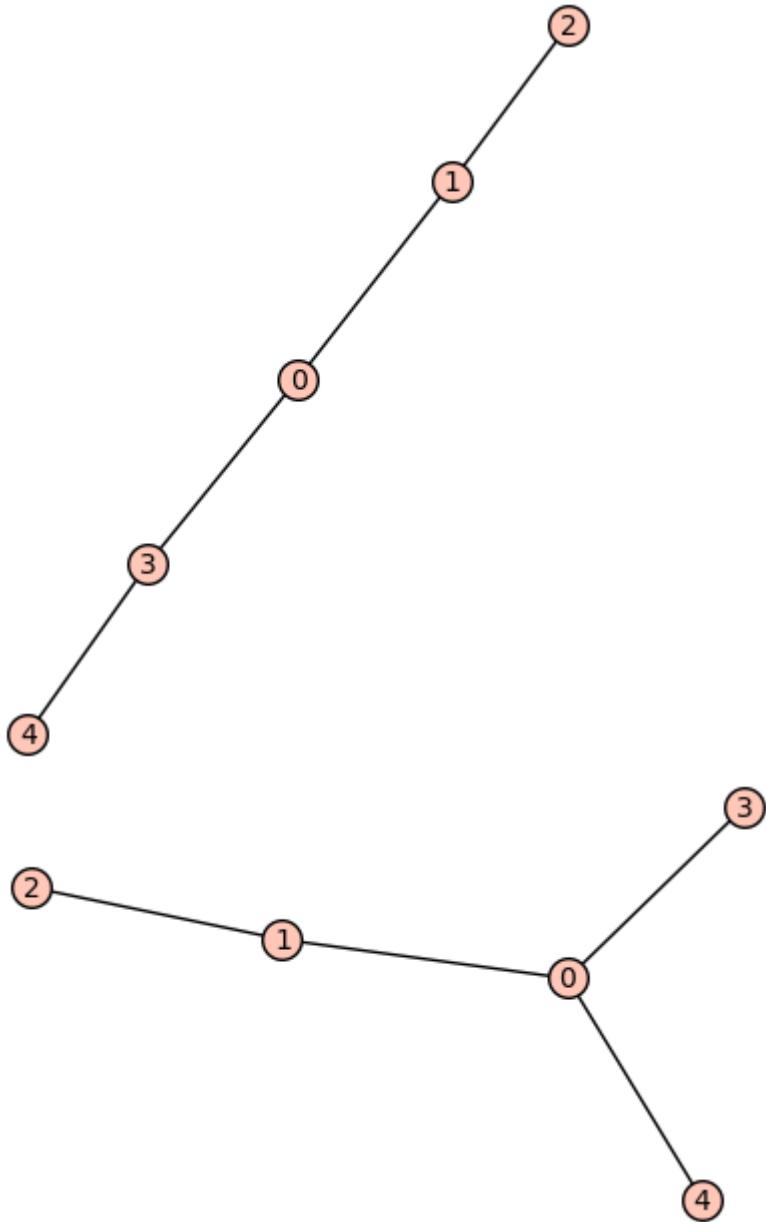
Bäume sind Graphen, die keine Kreise enthalten, also keine Pfade, die im selben Knoten beginnen und enden.

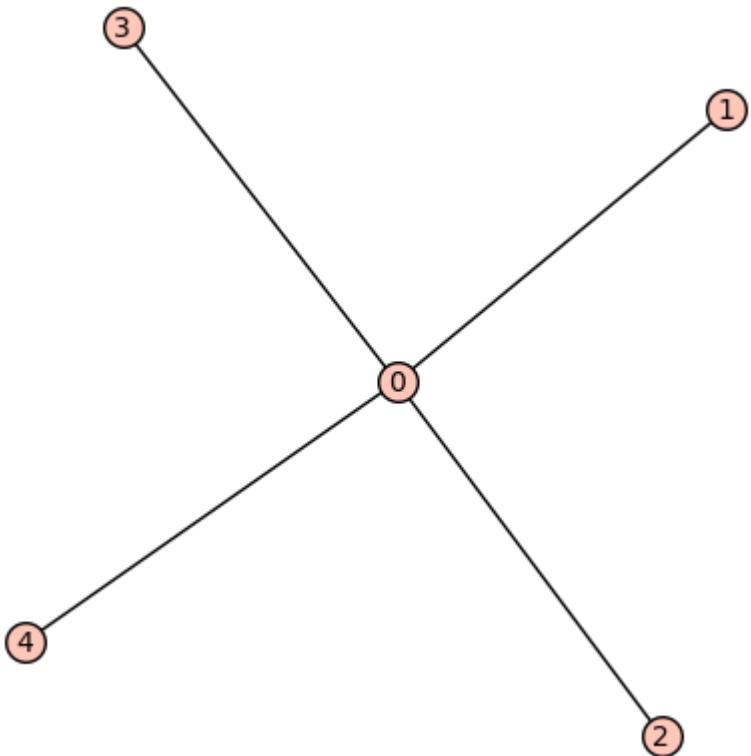
Wir erzeugen hier alle Bäume mit 5 Knoten.

```
In [97]: t5=graphs.trees(5)  
t5
```

```
Out[97]: <sage.graphs.trees.TreeIterator object at 0x7f7e2e1eedf0>
```

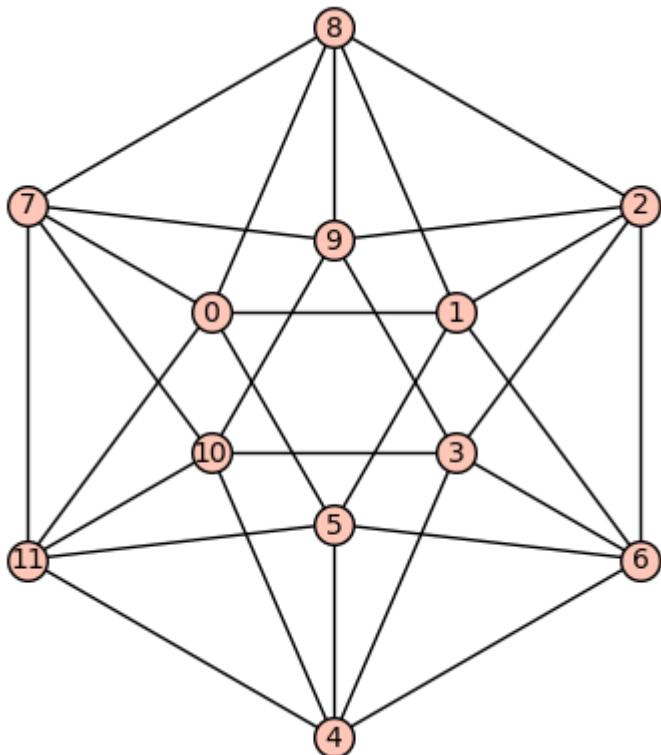
```
In [98]: for t in t5:  
    t.show()
```





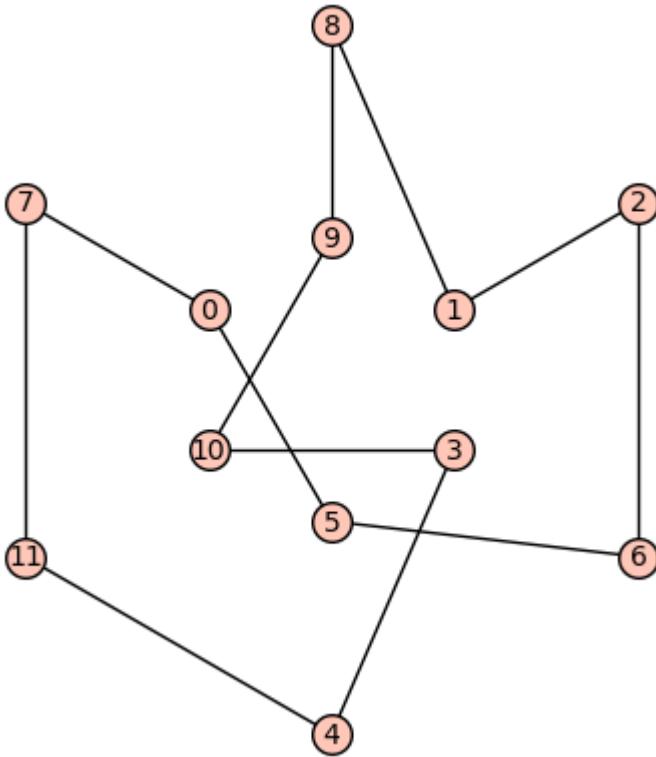
Der Graph bestehend aus Ecken und Kanten eines Ikosaeders.

```
In [99]: g20=graphs.IcosahedralGraph()
g20.show()
```



Hamiltonscher Kreis: Ein Kreis, der alle Knoten genau einmal enthält.

```
In [100...]: g20.hamiltonian_cycle().show()
```



Eulertour: ein Kreis, der alle Kanten genau einmal durchläuft.

```
In [101... g20.eulerian_circuit()
```

```
Out[101... False
```

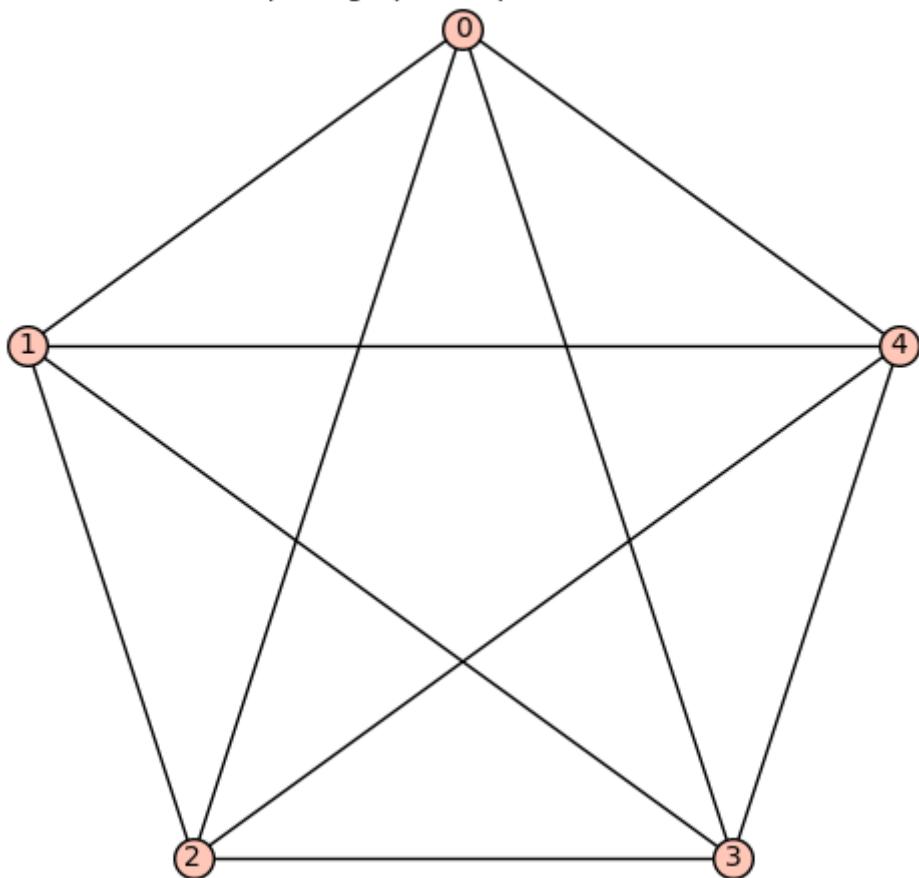
Der Ikosaedergraph hat also keine Eulertour.

Der vollständige Graph mit 5 Knoten. (D.h. alle möglichen Kanten existieren in dem Graphen)

```
In [102... k5=graphs.CompleteGraph(5)
k5
```

```
Out[102...]
```

Complete graph: Graph on 5 vertices



```
In [103... k5.eulerian_circuit()
```

```
Out[103... [(0, 4, None),  
 (4, 3, None),  
 (3, 2, None),  
 (2, 4, None),  
 (4, 1, None),  
 (1, 3, None),  
 (3, 0, None),  
 (0, 2, None),  
 (2, 1, None),  
 (1, 0, None)]
```

Darstellung der Eulertour im Bild des Graphen

```
In [104... for e,i in zip(k5.eulerian_circuit(),(1..)): k5.set_edge_label(e[0],e[1],i)
```

```
In [105... k5.show(edge_labels=True)
```

