6. Vorlesung 24.01.2025

Fehlerbehandlung: Exceptions

Wir nehmen verschiedene Fehlermeldungen unter die Lupe.

Ein **NameError** tritt immer bei undefinierten Variablen auf.

In [1]:
```
y
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call las
Cell In [1], line 1
----> 1 y

NameError: name 'y' is not defined
```

Ein **ValueError** besagt, daß zwar ein Objekt des richtigen Typs übergeben wurde, das aber einen ungeeigneten Wert hat.

In [2]:
```
factorial(-1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call las
Cell In [2], line 1
----> 1 factorial(-Integer(1))

File /usr/lib/python3/dist-packages/sage/symbolic/function.pyx:1033, in s
e.symbolic.function.BuiltinFunction.__call__ (build/cythonized/sage/symbo
c/function.c:11037)()
   1031 res = self._evalf_try_(*args)
   1032 if res is None:
-> 1033     res = super(BuiltinFunction, self).__call__(
   1034             *args, coerce=coerce, hold=hold)
   1035

File /usr/lib/python3/dist-packages/sage/symbolic/function.pyx:547, in sa
.symbolic.function.Function.__call__ (build/cythonized/sage/symbolic/func
on.c:6229)()
    545             raise TypeError("arguments must be symbolic expressio
s")
    546
--> 547 return call_registered_function(self._serial, self._nargs, args,
ld,
    548                                 not symbolic_input, SR)
    549

File /usr/lib/python3/dist-packages/sage/symbolic/pynac_function_impl.px
i:1, in sage.symbolic.expression.call_registered_function (build/cythoniz
d/sage/symbolic/expression.cpp:110612)()
----> 1 cpdef call_registered_function(unsigned serial,
      2                                int nargs,
      3                                list args,

File /usr/lib/python3/dist-packages/sage/symbolic/pynac_function_impl.pxi
9, in sage.symbolic.expression.call_registered_function (build/cythonized
age/symbolic/expression.cpp:110257)()
     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 /usr/lib/python3/dist-packages/sage/functions/other.py:1499, in Func
on_factorial._eval_(self, x)
   1497 if isinstance(x, Integer):
   1498     try:
-> 1499         return x.factorial()
   1500     except OverflowError:
   1501         return

File /usr/lib/python3/dist-packages/sage/rings/integer.pyx:4445, in sage.
ngs.integer.Integer.factorial (build/cythonized/sage/rings/integer.c:2921
()
   4443 """
   4444 if mpz_sgn(self.value) < 0:
-> 4445     raise ValueError("factorial only defined for non-negative int
ers")
   4446
   4447 if not mpz_fits_ulong_p(self.value):

ValueError: factorial only defined for non-negative integers
```

In [3]:
```
1/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call las
Cell In [3], line 1
----> 1 Integer(1)/Integer(0)

File /usr/lib/python3/dist-packages/sage/rings/integer.pyx:1987, in sage.
ngs.integer.Integer.__truediv__ (build/cythonized/sage/rings/integer.c:13
2)()
   1985 if type(left) is type(right):
   1986     if mpz_sgn((<Integer>right).value) == 0:
-> 1987         raise ZeroDivisionError("rational division by zero")
   1988     x = <Rational> Rational.__new__(Rational)
   1989     mpq_div_zz(x.value, (<Integer>left).value, (<Integer>right).v
ue)

ZeroDivisionError: rational division by zero
```

In [4]:
```
1/[1,2]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
Cell In [4], line 1
----> 1 Integer(1)/[Integer(1),Integer(2)]

File /usr/lib/python3/dist-packages/sage/rings/integer.pyx:2002, in sage.
ngs.integer.Integer.__truediv__ (build/cythonized/sage/rings/integer.c:13
8)()
   2000         return y
   2001
-> 2002     return coercion_model.bin_op(left, right, operator.truediv)
   2003
   2004 cpdef _div_(self, right):

File /usr/lib/python3/dist-packages/sage/structure/coerce.pyx:1248, in sa
.structure.coerce.CoercionModel.bin_op (build/cythonized/sage/structure/c
rce.c:11885)()
   1246     # We should really include the underlying error.
   1247     # This causes so much headache.
-> 1248     raise bin_op_exception(op, x, y)
   1249
   1250 cpdef canonical_coercion(self, x, y):

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

Wir wollen eine entsprechende Fehlermeldung in die folgende Funktion einbauen (Faktorielle).

In [5]:
```
def fac(n):
    if n == 0:
        return 1
    return n*fac(n-1)
```

In [6]:
```
fac(5)
```

Out[6]:
```
120
```

Die Funktion ist nicht gegen unendliche Rekursion gesichert.

In [7]:
```
fac(-1)
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call las
Cell In [7], line 1
----> 1 fac(-Integer(1))

Cell In [5], line 4, in fac(n)
      2 if n == Integer(0):
      3     return Integer(1)
----> 4 return n*fac(n-Integer(1))

Cell In [5], 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 [5], line 4, in fac(n)
      2 if n == Integer(0):
      3     return Integer(1)
----> 4 return n*fac(n-Integer(1))

Cell In [5], 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 o
ect
```

Eine Möglichkeit ist, einen String mit der entsprechenden Fehlermeldung zurückzugeben.

In [8]:
```
def fac(n):
    if n < 0:
        return "n nicht positiv"
    if n == 0:
        return 1
    return n*fac(n-1)
```

In [9]:
```
fac(-1)
```

Out[9]:
```
'n nicht positiv'
```

In [10]:
```
m=fac(-1)
```

In [11]:
```
m^2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
Cell In [11], line 1
----> 1 m**Integer(2)

File /usr/lib/python3/dist-packages/sage/rings/integer.pyx:2155, in sage.
ngs.integer.Integer.__pow__ (build/cythonized/sage/rings/integer.c:14654)
   2153         return coercion_model.bin_op(left, right, operator.pow)
   2154     # left is a non-Element: do the powering with a Python int
-> 2155     return left ** int(right)
   2156
   2157 cpdef _pow_(self, other):

TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

Besser ist es aber, eine richtige Fehlermeldung zu erzeugen, d.h., eine sogenannte "Exception". Dabei wird nicht einfach abgebrochen, sondern eine "Ausnahmesituation" erzeugt und gleichzeitig übermittelt, um welche Art von "Ausnahmesituation" es sich handelt. Die entsprechende Anweisung **raise** ist Teil der Programmiersprache python. In unserem Fall soll es analog zum **factorial**-Beispiel oben ein **ValueError** sein.

In [12]:
```
def fac(n):
    if n < 0:
        raise ValueError("n nicht positiv")
    if n == 0:
        return 1
    return n*fac(n-1)
```

In [13]:
```
fac(-1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call las
Cell In [13], line 1
----> 1 fac(-Integer(1))

Cell In [12], line 3, in fac(n)
      1 def fac(n):
      2     if n < Integer(0):
----> 3         raise ValueError("n nicht positiv")
      4     if n == Integer(0):
      5         return Integer(1)

ValueError: n nicht positiv
```

Exceptions haben den Vorteil, daß das Programm beim Auftreten eines Fehlers nicht einfach abbricht, sondern man die Möglichkeit hat, für die Ausnahmesituation vorzusorgen. Man kann ausprobieren, ob eine aufgerufene Funktion fehlerlos abläuft, und danach entsprechend der Art des Fehlers vorgehen. Die kritische Funktion wird mit **try** ausprobiert, und dann die verschiedenen Fehler mit **except** aufgefangen.

In [14]:
```
def tor(n): # wenn n <0, dann fac(-n)
    try:
        print ("trying")
        return fac(n)
        print ("succeeded")
    except ValueError:
        print ("failed")
        return fac(-n)
```

In [15]:
```
tor(1)
```

Out[15]:
```
trying
1
```

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

Out[16]:
```
trying
failed
1
```

Die Anweisung **print "succeeded"** wird nie erreicht.
Man kann auch verschiedene Exceptions nacheinander abfragen. Bei der Division haben wir zwei Arten von Exceptions gesehen.

In [17]:
```
1/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call las
Cell In [17], line 1
----> 1 Integer(1)/Integer(0)

File /usr/lib/python3/dist-packages/sage/rings/integer.pyx:1987, in sage.
ngs.integer.Integer.__truediv__ (build/cythonized/sage/rings/integer.c:13
2)()
   1985 if type(left) is type(right):
   1986     if mpz_sgn((<Integer>right).value) == 0:
-> 1987         raise ZeroDivisionError("rational division by zero")
   1988     x = <Rational> Rational.__new__(Rational)
   1989     mpq_div_zz(x.value, (<Integer>left).value, (<Integer>right).v
ue)

ZeroDivisionError: rational division by zero
```

In [18]:
```
1/[1,2]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
Cell In [18], line 1
----> 1 Integer(1)/[Integer(1),Integer(2)]

File /usr/lib/python3/dist-packages/sage/rings/integer.pyx:2002, in sage.
ngs.integer.Integer.__truediv__ (build/cythonized/sage/rings/integer.c:13
8)()
   2000         return y
   2001
-> 2002     return coercion_model.bin_op(left, right, operator.truediv)
   2003
   2004 cpdef _div_(self, right):

File /usr/lib/python3/dist-packages/sage/structure/coerce.pyx:1248, in sa
.structure.coerce.CoercionModel.bin_op (build/cythonized/sage/structure/c
rce.c:11885)()
   1246     # We should really include the underlying error.
   1247     # This causes so much headache.
-> 1248     raise bin_op_exception(op, x, y)
   1249
   1250 cpdef canonical_coercion(self, x, y):

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

Während Division durch eine Liste keinen Sinn ergibt, könnte man Division durch 0 als unendlich interpretieren:

In [19]:
```
def div(n,m):
    try:
        return n/m
    except ZeroDivisionError:
        return oo
    except TypeError:
        print ("geht nicht")
```

In [20]:
```
div(1,0)
```

Out[20]:
```
+Infinity
```

In [21]:
```
div(1,[1,2])
```

```
geht nicht
```

Programmierung: eigene Klassen

Wir definieren eine einfache Klasse, die die Arithmetik Max-Plus-Algebra implementiert.

Zunächst betrachten wir, wie binäre Operationen wie "+" technisch umgesetzt sind.

In [22]:
```
Q3=QQ^3
Q3
```

Out[22]:
```
Vector space of dimension 3 over Rational Field
```

In [23]:
```
u = Q3([1,-1,3])
u
```

Out[23]:
```
(1, -1, 3)
```

In [24]:
```
v = Q3([0,1,0])
v
```

Out[24]:
```
(0, 1, 0)
```

In [25]:
```
u+v
```

Out[25]:
```
(1, 0, 3)
```

Intern ist die Addition als Methode *add* implementiert.

In [26]:
```
u._add_(v)
```

Out[26]:
```
(1, 0, 3)
```

In [27]:
```
u._add_
```

Out[27]:
```
<built-in method _add_ of sage.modules.vector_rational_dense.Vector_ratio
nal_dense object at 0x7f325a730090>
```

Ein neuer Typ wird als sogenannte *Klasse* implementiert. Die Parameter eines Objekts werden in verschiedenen zugeordneten Variablen abgespeichert, hier genügt eine einzige Variable **val**. Das gerade bearbeitete Objekt ist unter **self** zugänglich. Zunächst brauchen wir die Methode **init** zur Erzeugung eines neuen Objekts sowie die Methode **repr** für die Ausgabe am Bildschirm.

In [28]:
```
class MaxPlus:
    def __init__(self, val):
        self.val = val
    def __repr__(self):
        return "["+str(self.val)+"]"
```

In [29]:
```
a = MaxPlus(5)
a
```

Out[29]:
```
[5]
```

In [30]:
```
parent(a)
```

Out[30]:
```
<class '__main__.MaxPlus'>
```

Als nächstes implementieren wir die Methoden für Addition und Multiplikation.

In [31]:
```
class MaxPlus:
    def __init__(self, val):
        self.val = val
    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 [32]:
```
a = MaxPlus(5)
a
```

Out[32]:
```
[5]
```

In [33]:
```
b = MaxPlus(-1)
```

In [34]:
```
a+b
```

Out[34]:
```
[5]
```

In [35]:
```
a*b
```

Out[35]:
```
[4]
```

Mehr geht nicht:

In [36]:
```
a-b
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
Cell In [36], line 1
----> 1 a-b

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

Das ganze gibt es schon.

In [37]:
```
TropicalSemiring??
```

```
<unknown>:191: DeprecationWarning: invalid escape sequence '\i'
```

In [38]:
```
MP=TropicalSemiring(QQ,use_min=False)
```

In [39]:
```
MP.zero()
```

Out[39]:
```
-infinity
```

In [40]:
```
MP.one()
```

Out[40]:
```
0
```

Vorsicht! Nicht alle Funktionen nehmen Rücksicht, dass die Null nicht 0 ist!

In [44]:
```
a = MP(-1)
b = MP(-2)
```

In [45]:
```
a+b
```

Out[45]:
```
-1
```

In [46]:
```
sum([a,b])
```

Out[46]:
```
0
```

In [47]:
```
sum([a,b],MP.zero())
```

Out[47]:
```
-1
```

Lineare Algebra mit Halbringen ist (noch) nicht implementiert

In [48]:
```
vector([MP(1),MP(2)])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
Cell In [48], line 1
----> 1 vector([MP(Integer(1)),MP(Integer(2))])

File /usr/lib/python3/dist-packages/sage/modules/free_module_element.pyx:
0, in sage.modules.free_module_element.vector (build/cythonized/sage/modu
s/free_module_element.c:6941)()
    578         sparse = False
    579
--> 580 v, R = prepare(v, R, degree)
    581
    582 M = FreeModule(R, len(v), bool(sparse))

File /usr/lib/python3/dist-packages/sage/modules/free_module_element.pyx:
1, in sage.modules.free_module_element.prepare (build/cythonized/sage/mod
es/free_module_element.c:7612)()
    679 ring = v.universe()
    680 if not is_Ring(ring):
--> 681     raise TypeError("unable to find a common ring for all element
s")
    682 return v, ring
    683

TypeError: unable to find a common ring for all elements
```

Abstrakte Algebra: Gruppen

sage kennt außer den bisher behandelten Polynomen und Körpern noch viele andere algebraische Strukturen. Als typisches Beispiel betrachten wir die Gruppe der Permutationen der Ordnung 5.

In [49]:
```
S5 = SymmetricGroup(5)
S5
```

Out[49]:
```
Symmetric group of order 5! as a permutation group
```

Elemente können direkt aus einer Liste erzeugt werden. Intern wird die Permutation sofort in ein Produkt von Zyklen zerlegt.

In [50]:
```
p1=S5 ( [2,1,4,5,3])
p1
```

Out[50]:
```
(1,2)(3,4,5)
```

Allerdings muß die Ordnung übereinstimmen.

In [51]:
```
p1=S5 ( [6,2,1,4,5,3])
p1
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call las
Cell In [51], line 1
----> 1 p1=S5 ( [Integer(6),Integer(2),Integer(1),Integer(4),Integer(5),I
eger(3)])
      2 p1

File /usr/lib/python3/dist-packages/sage/structure/parent.pyx:898, in sag
structure.parent.Parent.__call__ (build/cythonized/sage/structure/paren
t.c:9522)()
    896 if mor is not None:
    897     if no_extra_args:
--> 898         return mor._call_(x)
    899     else:
    900         return mor._call_with_args(x, args, kwds)

File /usr/lib/python3/dist-packages/sage/structure/coerce_maps.pyx:161, i
sage.structure.coerce_maps.DefaultConvertMap_unique._call_ (build/cythoni
d/sage/structure/coerce_maps.c:4799)()
    159             print(type(C), C)
    160             print(type(C._element_constructor), C._element_constr
tor)
--> 161         raise
    162
    163 cpdef Element _call_with_args(self, x, args=(), kwds={}):

File /usr/lib/python3/dist-packages/sage/structure/coerce_maps.pyx:156, i
sage.structure.coerce_maps.DefaultConvertMap_unique._call_ (build/cythoni
d/sage/structure/coerce_maps.c:4691)()
    154 cdef Parent C = self._codomain
    155 try:
--> 156     return C._element_constructor(x)
    157 except Exception:
    158     if print_warnings:

File /usr/lib/python3/dist-packages/sage/groups/perm_gps/permgroup.py:831
in PermutationGroup_generic._element_constructor_(self, x, check)
    827     if compatible_domains and (isinstance(self, SymmetricGroup)
    828                                or x.gap() in self.gap()):
    829         return self.element_class(x, self, check=False)
--> 831 return self.element_class(x, self, check=check)

File /usr/lib/python3/dist-packages/sage/groups/perm_gps/permgroup_elemen
t.pyx:465, in sage.groups.perm_gps.permgroup_element.PermutationGroupElem
t.__init__ (build/cythonized/sage/groups/perm_gps/permgroup_element.c:558
()
    463         self._set_list_cycles(g, convert)
    464     else:
--> 465         self._set_list_images(g, convert)
    466 elif isinstance(g, str):
    467     self._set_string(g)

File /usr/lib/python3/dist-packages/sage/groups/perm_gps/permgroup_elemen
t.pyx:553, in sage.groups.perm_gps.permgroup_element.PermutationGroupElem
t._set_list_images (build/cythonized/sage/groups/perm_gps/permgroup_eleme
t.c:6644)()
    551 """
    552 cdef int i, j, vn = len(v)
--> 553 assert(vn <= self.n)
    554 if convert:
    555     convert_dict = self._parent._domain_to_gap
```

```
AssertionError:
```

Die Permutation kann auch direkt als Zyklus eingegeben werden.

In [52]:
```
p2= S5( (2,3))
p2
```

Out[52]:
```
(2,3)
```

Dann stehen alle Gruppenoperationen zur Verfügung.

In [53]:
```
p1*p2
```

Out[53]:
```
(1,3,4,5,2)
```

In [54]:
```
p2*p1
```

Out[54]:
```
(1,2,4,5,3)
```

In [55]:
```
p1^-1
```

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

Analog zur linearen Hülle in der linearen Algebra ist die von einer Familie *erzeugte Untergruppe* definiert als die kleinste Gruppe, die die besagte Familie enthält.

In [56]:
```
G = S5.subgroup([p1,p2])
G
```

Out[56]:
```
Subgroup generated by [(2,3), (1,2)(3,4,5)] of (Symmetric group of order
5! as a permutation group)
```

Die Liste der Elemente.

In [57]:
```
len(G.list())
```

Out[57]:
```
120
```

Die Verknüpfungstafel. Die Zeilen und Spalten sind entsprechend der obigen Liste duchnumeriert.

In [58]:
```
G.cayley_table()
```

Out[58]:

```
 *  aa ab ac ad ae af ag ah ai aj ak al am an ao ap aq ar as at au av aw
ep

+--------------------------------------------------------------------
aa| aa ab ac ad ae af ag ah ai aj ak al am an ao ap aq ar as at au av aw
ep
ab| ab aa ad ac af ae ah ag aj ai al ak an am ap ao ar aq at as av au ax
eo
ac| ac ae aa af ab ad ai ak ag al ah aj ao aq am ar an ap au aw as ax at
en
ad| ad af ab ae aa ac aj al ah ak ag ai ap ar an aq am ao av ax at aw as
em
ae| ae ac af aa ad ab ak ai al ag aj ah aq ao ar am ap an aw au ax as av
el
af| af ad ae ab ac aa al aj ak ah ai ag ar ap aq an ao am ax av aw at au
ek
ag| ag ah am an as at aa ab ao ap au av ac ad ai aj aw ax ae af ak al aq
ej
ah| ah ag an am at as ab aa ap ao av au ad ac aj ai ax aw af ae al ak ar
ei
ai| ai ak ao aq au aw ac ae am ar as ax aa af ag al at av ab ad ah aj an
eh
aj| aj al ap ar av ax ad af an aq at aw ab ae ah ak as au aa ac ag ai am
eg
ak| ak ai aq ao aw au ae ac ar am ax as af aa al ag av at ad ab aj ah ap
ef
al| al aj ar ap ax av af ad aq an aw at ae ab ak ah au as ac aa ai ag ao
ee
am| am as ag at ah an ao au aa av ab ap ai aw ac ax ad aj ak aq ae ar af
ed
an| an at ah as ag am ap av ab au aa ao aj ax ad aw ac ai al ar af aq ae
ec
ao| ao au ai aw ak aq am as ac ax ae ar ag at aa av af al ah an ab ap ad
eb
ap| ap av aj ax al ar an at ad aw af aq ah as ab au ae ak ag am aa ao ac
ea
aq| aq aw ak au ai ao ar ax ae as ac am al av af at aa ag aj ap ad an ab
dz
ar| ar ax al av aj ap aq aw af at ad an ak au ae as ab ah ai ao ac am aa
dy
as| as am at ag an ah au ao av aa ap ab aw ai ax ac aj ad aq ak ar ae al
dx
at| at an as ah am ag av ap au ab ao aa ax aj aw ad ai ac ar al aq af ak
dw
au| au ao aw ai aq ak as am ax ac ar ae at ag av aa al af an ah ap ab aj
dv
av| av ap ax aj ar al at an aw ad aq af as ah au ab ak ae am ag ao aa ai
du
aw| aw aq au ak ao ai ax ar as ae am ac av al at af ag aa ap aj an ad ah
dt
ax| ax ar av al ap aj aw aq at af an ad au ak as ae ah ab ao ai am ac ag
ds
ay| ay az ba bb bc bd bw bx by bz ca cb cu cv cw cx cy cz ds dt du dv dw
dr
az| az ay bb ba bd bc bx bw bz by cb ca cv cu cx cw cz cy dt ds dv du dx
dq
ba| ba bc ay bd az bb by ca bw cb bx bz cw cy cu cz cv cx du dw ds dx dt
dp
bb| bb bd az bc ay ba bz cb bx ca bw by cx cz cv cy cu cw dv dx dt dw ds
do
bc| bc ba bd ay bb az ca by cb bw bz bx cy cw cz cu cx cv dw du dx ds dv
dn
bd| bd bb bc az ba ay cb bz ca bx by bw cz cx cy cv cw cu dx dv dw dt du
```

```
dm
be| be bf bk bl bq br cc cd ci cj co cp da db dg dh dm dn dy dz ee ef ek
dl
bf| bf be bl bk br bq cd cc cj ci cp co db da dh dg dn dm dz dy ef ee el
dk
bg| bg bi bm bo bs bu ce cg ck cm cq cs dc de di dk do dq ea ec eg ei em
dj
bh| bh bj bn bp bt bv cf ch cl cn cr ct dd df dj dl dp dr eb ed eh ej en
di
bi| bi bg bo bm bu bs cg ce cm ck cs cq de dc dk di dq do ec ea ei eg eo
dh
bj| bj bh bp bn bv bt ch cf cn cl ct cr df dd dl dj dr dp ed eb ej eh ep
dg
bk| bk bq be br bf bl ci co cc cp cd cj dg dm da dn db dh ee ek dy el dz
df
bl| bl br bf bq be bk cj cp cd co cc ci dh dn db dm da dg ef el dz ek dy
de
bm| bm bs bg bu bi bo ck cq ce cs cg cm di do dc dq de dk eg em ea eo ec
dd
bn| bn bt bh bv bj bp cl cr cf ct ch cn dj dp dd dr df dl eh en eb ep ed
dc
bo| bo bu bi bs bg bm cm cs cg cq ce ck dk dq de do dc di ei eo ec em ea
db
bp| bp bv bj bt bh bn cn ct ch cr cf cl dl dr df dp dd dj ej ep ed en eb
da
bq| bq bk br be bl bf co ci cp cc cj cd dm dg dn da dh db ek ee el dy ef
cz
br| br bl bq bf bk be cp cj co cd ci cc dn dh dm db dg da el ef ek dz ee
cy
bs| bs bm bu bg bo bi cq ck cs ce cm cg do di dq dc dk de em eg eo ea ei
cx
bt| bt bn bv bh bp bj cr cl ct cf cn ch dp dj dr dd dl df en eh ep eb ej
cw
bu| bu bo bs bi bm bg cs cm cq cg ck ce dq dk do de di dc eo ei em ec eg
cv
bv| bv bp bt bj bn bh ct cn cr ch cl cf dr dl dp df dj dd ep ej en ed eh
cu
bw| bw bx cu cv ds dt ay az cw cx du dv ba bb by bz dw dx bc bd ca cb cy
ct
bx| bx bw cv cu dt ds az ay cx cw dv du bb ba bz by dx dw bd bc cb ca cz
cs
by| by ca cw cy du dw ba bc cu cz ds dx ay bd bw cb dt dv az bb bx bz cv
cr
bz| bz cb cx cz dv dx bb bd cv cy dt dw az bc bx ca ds du ay ba bw by cu
cq
ca| ca by cy cw dw du bc ba cz cu dx ds bd ay cb bw dv dt bb az bz bx cx
cp
cb| cb bz cz cx dx dv bd bb cy cv dw dt bc az ca bx du ds ba ay by bw cw
co
cc| cc cd da db dy dz be bf dg dh ee ef bk bl ci cj ek el bq br co cp dm
cn
cd| cd cc db da dz dy bf be dh dg ef ee bl bk cj ci el ek br bq cp co dn
cm
ce| ce cg dc de ea ec bg bi di dk eg ei bm bo ck cm em eo bs bu cq cs do
cl
cf| cf ch dd df eb ed bh bj dj dl eh ej bn bp cl cn en ep bt bv cr ct dp
ck
cg| cg ce de dc ec ea bi bg dk di ei eg bo bm cm ck eo em bu bs cs cq dq
cj
ch| ch cf df dd ed eb bj bh dl dj ej eh bp bn cn cl ep en bv bt ct cr dr
ci
ci| ci co dg dm ee ek bk bq da dn dy el be br cc cp dz ef bf bl cd cj db
ch
```

```
cj| cj cp dh dn ef el bl br db dm dz ek bf bq cd co dy ee be bk cc ci da
cg
ck| ck cq di do eg em bm bs dc dq ea eo bg bu ce cs ec ei bi bo cg cm de
cf
cl| cl cr dj dp eh en bn bt dd dr eb ep bh bv cf ct ed ej bj bp ch cn df
ce
cm| cm cs dk dq ei eo bo bu de do ec em bi bs cg cq ea eg bg bm ce ck dc
cd
cn| cn ct dl dr ej ep bp bv df dp ed en bj bt ch cr eb eh bh bn cf cl dd
cc
co| co ci dm dg ek ee bq bk dn da el dy br be cp cc ef dz bl bf cj cd dh
cb
cp| cp cj dn dh el ef br bl dm db ek dz bq bf co cd ee dy bk be ci cc dg
ca
cq| cq ck do di em eg bs bm dq dc eo ea bu bg cs ce ei ec bo bi cm cg dk
bz
cr| cr cl dp dj en eh bt bn dr dd ep eb bv bh ct cf ej ed bp bj cn ch dl
by
cs| cs cm dq dk eo ei bu bo do de em ec bs bi cq cg eg ea bm bg ck ce di
bx
ct| ct cn dr dl ep ej bv bp dp df en ed bt bj cr ch eh eb bn bh cl cf dj
bw
cu| cu ds bw dt bx cv cw du ay dv az cx by dw ba dx bb bz ca cy bc cz bd
bv
cv| cv dt bx ds bw cu cx dv az du ay cw bz dx bb dw ba by cb cz bd cy bc
bu
cw| cw du by dw ca cy cu ds ba dx bc cz bw dt ay dv bd cb bx cv az cx bb
bt
cx| cx dv bz dx cb cz cv dt bb dw bd cy bx ds az du bc ca bw cu ay cw ba
bs
cy| cy dw ca du by cw cz dx bc ds ba cu cb dv bd dt ay bw bz cx bb cv az
br
cz| cz dx cb dv bz cx cy dw bd dt bb cv ca du bc ds az bx by cw ba cu ay
bq
da| da dy cc dz cd db dg ee be ef bf dh ci ek bk el bl cj co dm bq dn br
bp
db| db dz cd dy cc da dh ef bf ee be dg cj el bl ek bk ci cp dn br dm bq
bo
dc| dc ea ce ec cg de di eg bg ei bi dk ck em bm eo bo cm cq do bs dq bu
bn
dd| dd eb cf ed ch df dj eh bh ej bj dl cl en bn ep bp cn cr dp bt dr bv
bm
de| de ec cg ea ce dc dk ei bi eg bg di cm eo bo em bm ck cs dq bu do bs
bl
df| df ed ch eb cf dd dl ej bj eh bh dj cn ep bp en bn cl ct dr bv dp bt
bk
dg| dg ee ci ek co dm da dy bk el bq dn cc dz be ef br cp cd db bf dh bl
bj
dh| dh ef cj el cp dn db dz bl ek br dm cd dy bf ee bq co cc da be dg bk
bi
di| di eg ck em cq do dc ea bm eo bs dq ce ec bg ei bu cs cg de bi dk bo
bh
dj| dj eh cl en cr dp dd eb bn ep bt dr cf ed bh ej bv ct ch df bj dl bp
bg
dk| dk ei cm eo cs dq de ec bo em bu do cg ea bi eg bs cq ce dc bg di bm
bf
dl| dl ej cn ep ct dr df ed bp en bv dp ch eb bj eh bt cr cf dd bh dj bn
be
dm| dm ek co ee ci dg dn el bq dy bk da cp ef br dz be cc cj dh bl db bf
bd
dn| dn el cp ef cj dh dm ek br dz bl db co ee bq dy bf cd ci dg bk da be
bc
do| do em cq eg ck di dq eo bs ea bm dc cs ei bu ec bg ce cm dk bo de bi
```

```
bb
dp| dp en cr eh cl dj dr ep bt eb bn dd ct ej bv ed bh cf cn dl bp df bj
ba
dq| dq eo cs ei cm dk do em bu ec bo de cq eg bs ea bi cg ck di bm dc bg
az
dr| dr ep ct ej cn dl dp en bv ed bp df cr eh bt eb bj ch cl dj bn dd bh
ay
ds| ds cu dt bw cv bx du cw dv ay cx az dw by dx ba bz bb cy ca cz bc cb
ax
dt| dt cv ds bx cu bw dv cx du az cw ay dx bz dw bb by ba cz cb cy bd ca
aw
du| du cw dw by cy ca ds cu dx ba cz bc dt bw dv ay cb bd cv bx cx az bz
av
dv| dv cx dx bz cz cb dt cv dw bb cy bd ds bx du az ca bc cu bw cw ay by
au
dw| dw cy du ca cw by dx cz ds bc cu ba dv cb dt bd bw ay cx bz cv bb bx
at
dx| dx cz dv cb cx bz dw cy dt bd cv bb du ca ds bc bx az cw by cu ba bw
as
dy| dy da dz cc db cd ee dg ef be dh bf ek ci el bk cj bl dm co dn bq cp
ar
dz| dz db dy cd da cc ef dh ee bf dg be el cj ek bl ci bk dn cp dm br co
aq
ea| ea dc ec ce de cg eg di ei bg dk bi em ck eo bm cm bo do cq dq bs cs
ap
eb| eb dd ed cf df ch eh dj ej bh dl bj en cl ep bn cn bp dp cr dr bt ct
ao
ec| ec de ea cg dc ce ei dk eg bi di bg eo cm em bo ck bm dq cs do bu cq
an
ed| ed df eb ch dd cf ej dl eh bj dj bh ep cn en bp cl bn dr ct dp bv cr
am
ee| ee dg ek ci dm co dy da el bk dn bq dz cc ef be cp br db cd dh bf cj
al
ef| ef dh el cj dn cp dz db ek bl dm br dy cd ee bf co bq da cc dg be ci
ak
eg| eg di em ck do cq ea dc eo bm dq bs ec ce ei bg cs bu de cg dk bi cm
aj
eh| eh dj en cl dp cr eb dd ep bn dr bt ed cf ej bh ct bv df ch dl bj cn
ai
ei| ei dk eo cm dq cs ec de em bo do bu ea cg eg bi cq bs dc ce di bg ck
ah
ej| ej dl ep cn dr ct ed df en bp dp bv eb ch eh bj cr bt dd cf dj bh cl
ag
ek| ek dm ee co dg ci el dn dy bq da bk ef cp dz br cc be dh cj db bl cd
af
el| el dn ef cp dh cj ek dm dz br db bl ee co dy bq cd bf dg ci da bk cc
ae
em| em do eg cq di ck eo dq ea bs dc bm ei cs ec bu ce bg dk cm de bo cg
ad
en| en dp eh cr dj cl ep dr eb bt dd bn ej ct ed bv cf bh dl cn df bp ch
ac
eo| eo dq ei cs dk cm em do ec bu de bo eg cq ea bs cg bi di ck dc bm ce
ab
ep| ep dr ej ct dl cn en dp ed bv df bp eh cr eb bt ch bj dj cl dd bn cf
aa
```

Es können verschiedene Eigenschaften der Untergruppe abgefragt werden.

In [59]:
```
G.is_abelian()
```

Out[59]:
```
False
```

Wieviele abelsche Untergruppen hat S5? Zunächst eine Liste aller Untergruppen.

In [60]:
```
S5ss = S5.subgroups()
len(S5ss)
```

Out[60]:
```
156
```

Aus dieser Liste wählen wir die abelschen aus.

In [61]:
```
SS5ab =[G for G in S5ss if G.is_abelian()]
len(SS5ab)
```

Out[61]:
```
87
```

Kombinatorik: Graphen

sage kennt viele Objekte aus der diskreten Mathematik, wir betrachten als stellvertretendes Beispiel endliche Graphen. Wir beginnen mit einem leeren Graphen:

In [62]:
```
g = Graph()
```

An den Graphen können nach und nach Knoten und Kanten angehängt werden. Dabei wird keine Kopie angelegt, sondern der Graph selbst verändert.

In [63]:
```
g.add_vertex(0)
g
```

Out[63]: Graph on 1 vertex

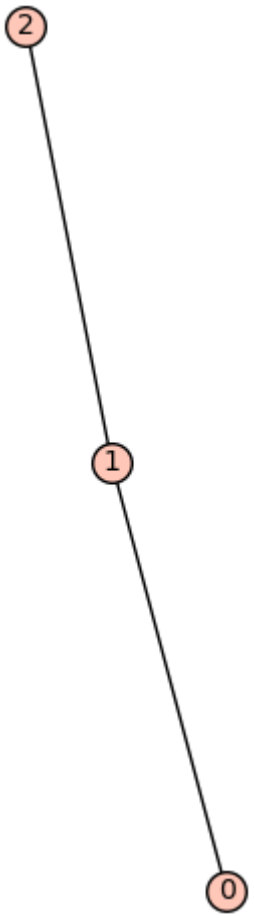
Ein Graph kann mit **.show()** oder **.plot()** angezeigt werden.

In [64]:
```
g.show()
```

Nicht vorhandene Knoten werden bei Bedarf automatisch erzeugt.

In [65]:
```
g.add_edge(1,2)
g.add_edge(1,0)
g.plot()
```

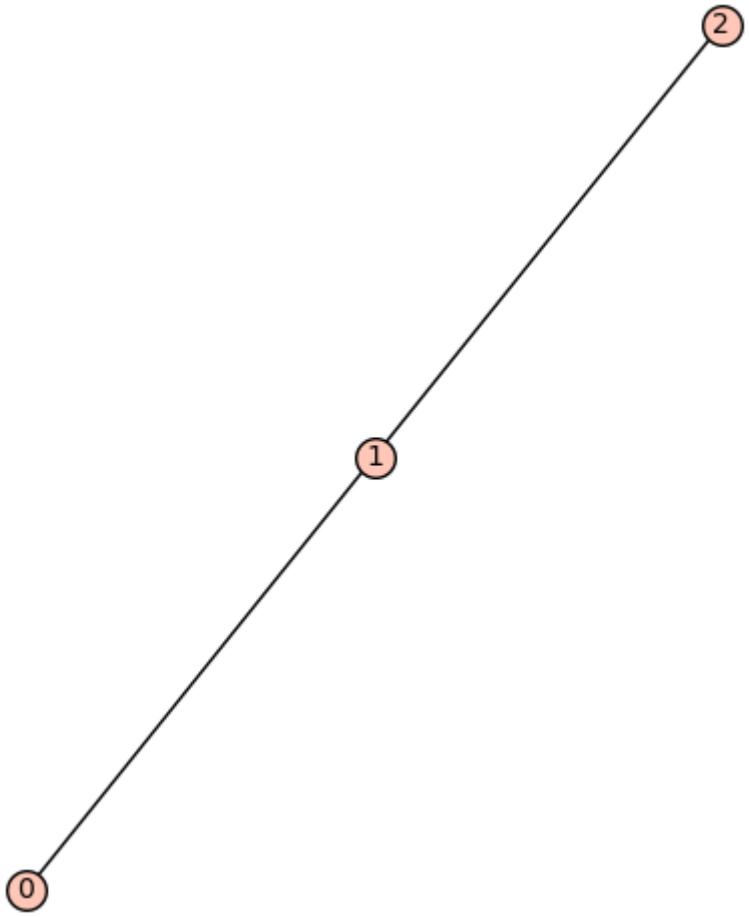
Out[65]:



schon vorhandene Kanten oder Knoten werden nicht verdoppelt.

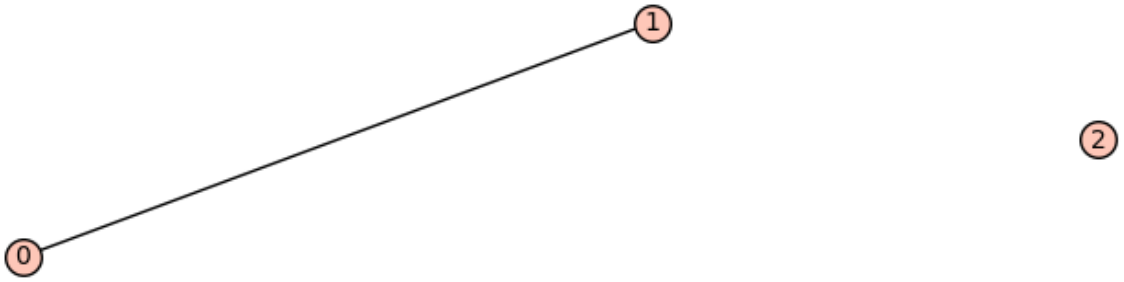
In [66]:
```
g.add_edge(1,2)
g.add_edge(1,2)
g.plot()
```

Out[66]:



In [67]:
```
g.delete_edge(1,2)
g.plot()
```

Out[67]:



Die Bezeichnungen der Knoten können beliebige Objekte sein, allerdings müssen sie aus technischen Gründen unveränderbar (immutable) sein:

In [68]:
```
a = matrix([[x,1],[1,2]])
g.add_edge(0,a)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
Cell In [68], line 2
      1 a = matrix([[x,Integer(1)],[Integer(1),Integer(2)]])
----> 2 g.add_edge(Integer(0),a)

File /usr/lib/python3/dist-packages/sage/graphs/generic_graph.py:10983, i
GenericGraph.add_edge(self, u, v, label)
  10980         except Exception:
  10981             pass
> 10983 self._backend.add_edge(u, v, label, self._directed)

File /usr/lib/python3/dist-packages/sage/graphs/base/c_graph.pyx:2359, in
age.graphs.base.c_graph.CGraphBackend.add_edge (build/cythonized/sage/gra
s/base/c_graph.cpp:16693)()
   2357         self.add_edge(u,v,l,directed)
   2358
-> 2359 cpdef add_edge(self, object u, object v, object l, bint directed)
   2360     """
   2361     Add the edge ``(u,v)`` to self.

File /usr/lib/python3/dist-packages/sage/graphs/base/c_graph.pyx:2435, in
age.graphs.base.c_graph.CGraphBackend.add_edge (build/cythonized/sage/gra
s/base/c_graph.cpp:16279)()
   2433
   2434         cdef int u_int = self.check_labelled_vertex(u, False)
-> 2435         cdef int v_int = self.check_labelled_vertex(v, False)
   2436
   2437         cdef CGraph cg = self.cg()

File /usr/lib/python3/dist-packages/sage/graphs/base/c_graph.pyx:1655, in
age.graphs.base.c_graph.CGraphBackend.check_labelled_vertex (build/cython
ed/sage/graphs/base/c_graph.cpp:11787)()
   1653 cdef CGraph G = self.cg()
   1654
-> 1655 cdef int u_int = self.get_vertex(u)
   1656 if u_int != -1:
   1657     if not bitset_in(G.active_vertices, u_int):

File /usr/lib/python3/dist-packages/sage/graphs/base/c_graph.pyx:1612, in
age.graphs.base.c_graph.CGraphBackend.get_vertex (build/cythonized/sage/g
phs/base/c_graph.cpp:11284)()
   1610 cdef CGraph G = self.cg()
   1611 cdef long u_long
-> 1612 if u in vertex_ints:
   1613     return vertex_ints[u]
   1614 try:

File /usr/lib/python3/dist-packages/sage/matrix/matrix0.pyx:5902, in sage
atrix.matrix0.Matrix.__hash__ (build/cythonized/sage/matrix/matrix0.c:396
6)()
   5900 """
   5901 if not self._is_immutable:
-> 5902     raise TypeError("mutable matrices are unhashable")
   5903 if self.hash != -1:
   5904     return self.hash

TypeError: mutable matrices are unhashable
```

In [69]:
```
a.set_immutable()
a[0,0] = 1
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call las
Cell In [69], line 2
      1 a.set_immutable()
----> 2 a[Integer(0),Integer(0)] = Integer(1)

File /usr/lib/python3/dist-packages/sage/matrix/matrix0.pyx:1391, in sage
atrix.matrix0.Matrix.__setitem__ (build/cythonized/sage/matrix/matrix0.c:
05)()
   1389 # If the matrix is immutable, check_mutability will raise an
   1390 # exception.
-> 1391 self.check_mutability()
   1392
   1393 if type(key) is tuple:

File /usr/lib/python3/dist-packages/sage/matrix/matrix0.pyx:388, in sage.
trix.matrix0.Matrix.check_mutability (build/cythonized/sage/matrix/matrix
0.c:5733)()
    386 """
    387 if self._is_immutable:
--> 388     raise ValueError("matrix is immutable; please change a copy i
tead (i.e., use copy(M) to change a copy of M).")
    389 else:
    390     self._cache = None

ValueError: matrix is immutable; please change a copy instead (i.e., use
py(M) to change a copy of M).
```

In [70]:
```
a
```

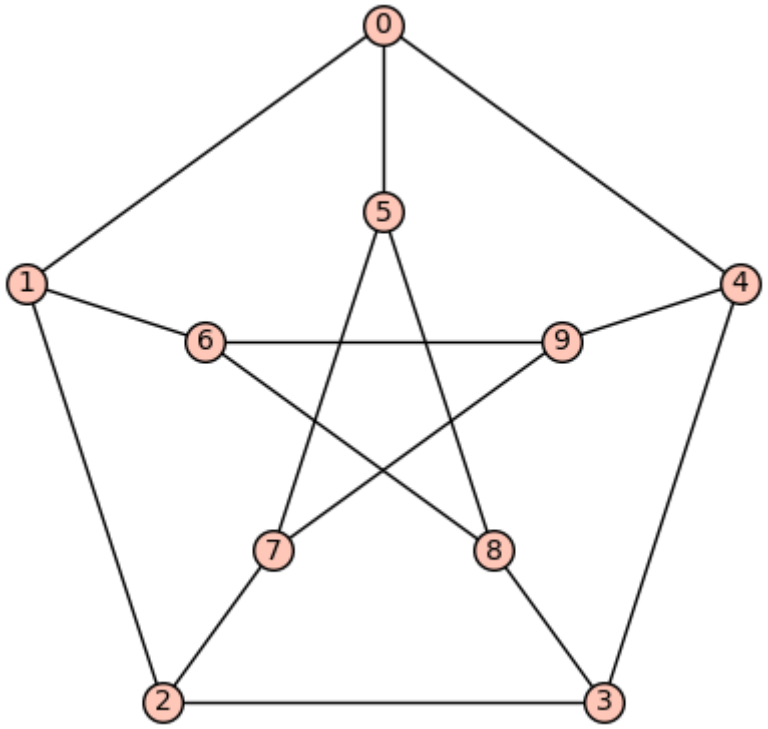
Out[70]:
```
[x 1]
[1 2]
```

In [71]:
```
g.add_edge(0,a)
g.show()
```

Die Bilder können auch in verschiedenen Formaten exportiert werden.

Eine ganze Reihe vordefinierter Graphen ist in der Bibliothek **graphs** zugänglich.

In [72]:
```
p = graphs.PetersenGraph()
g=p.show()
```



Es stehen viele Algorithmen zur Verfügung.

In [73]:
```
p.is_planar()
```

Out[73]:
```
False
```

In [74]:
```
p.neighbors(1)
```

Out[74]:
```
[0, 2, 6]
```

In [75]:
```
p.distance(1,4)
```

Out[75]:
```
2
```

Eine *Färbung* eines Graphen ist eine Abbildung, die jedem Knoten eine Farbe zuordnet, und zwar so, daß benachbarte Knoten verschiedene Farben haben. Die *chromatische Zahl* ist die kleinstmögliche Anzahl von Farben, mit denen eine Färbung möglich ist.

In [76]:
```
p.chromatic_number()
```

Out[76]:
```
3
```

Eine solche minimale Färbung kann explizit gefunden werden.

In [77]:
```
p.coloring()
```

Out[77]:
```
[[0, 2, 6], [1, 3, 5, 9], [4, 7, 8]]
```

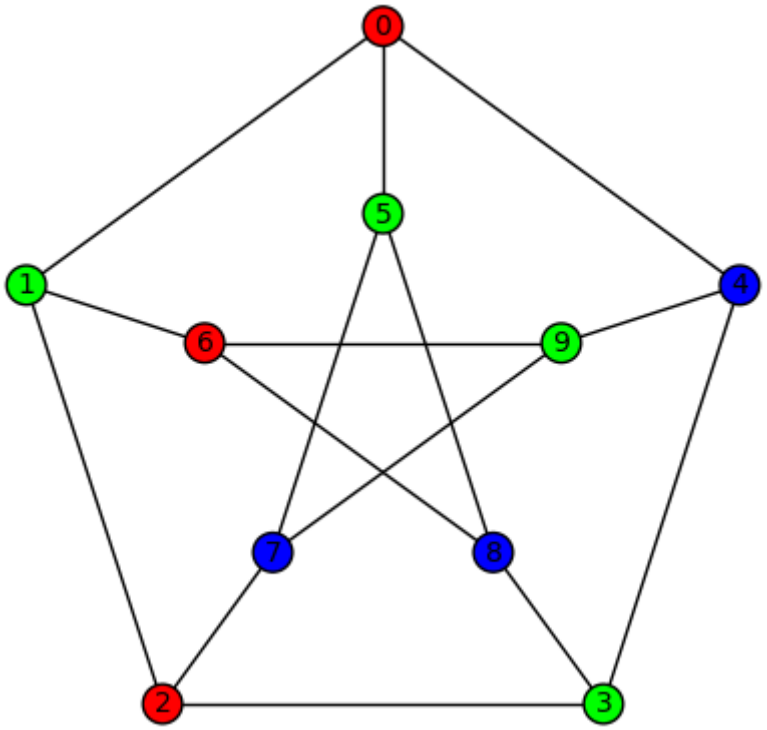
Um die Färbung zu zeichnen, kann man auch direkt Farbwerte erzeugen.

In [78]:
```
c=p.coloring(hex_colors=True)
c
```

Out[78]:
```
{'#ff0000': [0, 2, 6], '#00ff00': [1, 3, 5, 9], '#0000ff': [4, 7, 8]}
```

Diese können der Methode **.show()** als Option übergeben werden.

In [79]:
```
p.show(vertex_colors=c)
```



Wir können die symmetrische Gruppe auf den Knoten des Graphen *agieren* lassen, d.h., die Knoten anhand von Permutationen umzubenennen.

In [80]:
```
S10 = SymmetricGroup(10)
```

Wir vertauschen die inneren mit den äußeren Knoten. Dabei ist zu beachten, daß die Knoten anhand ihrer Position in der Knotenliste identifiziert werden.

In [81]:
```
u=S10([6,7,8,9,10,1,2,3,4,5])
u
```

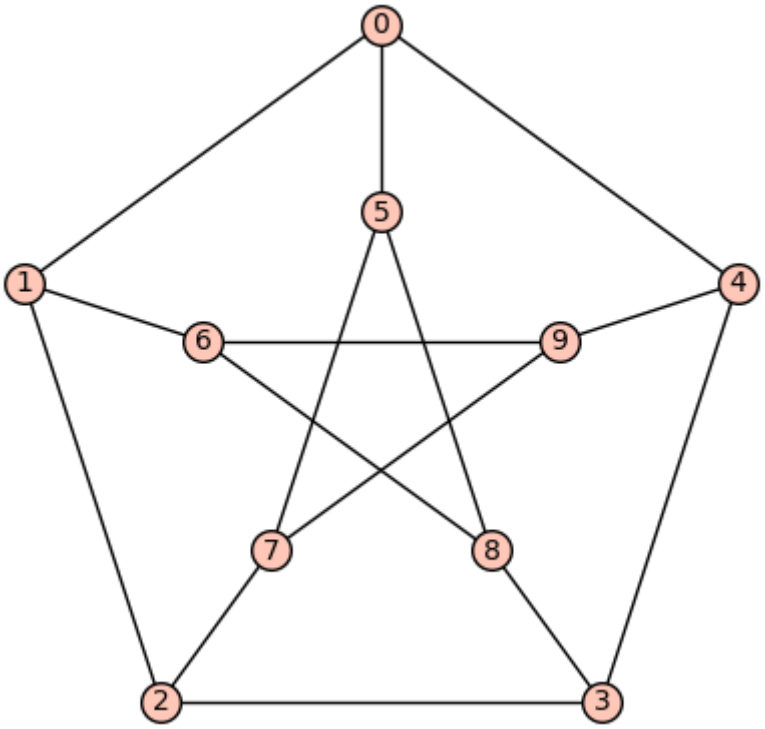
Out[81]:
```
(1,6)(2,7)(3,8)(4,9)(5,10)
```

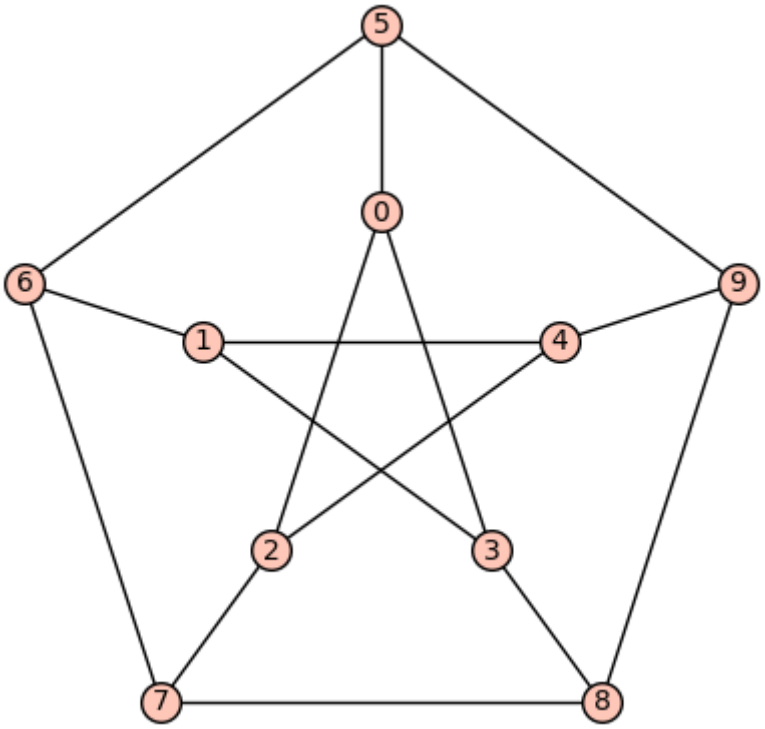
da der Graph direkt verändert wird, legen wir zuerst Kopien an.

In [82]:
```
p1=p.copy()
p2=p.copy()
```

In [83]:
```
p2.relabel(u)
```

In [84]:
```
p1.show()
p2.show()
```





In [85]:
```
p1.is_isomorphic(p2)
```

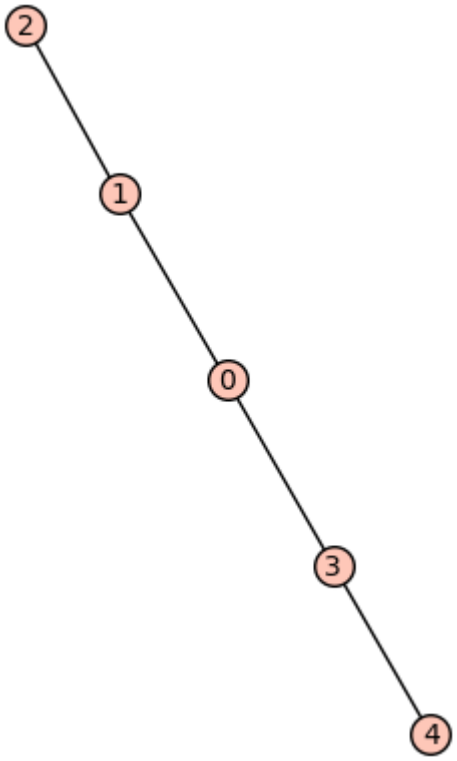
Out[85]:
```
True
```

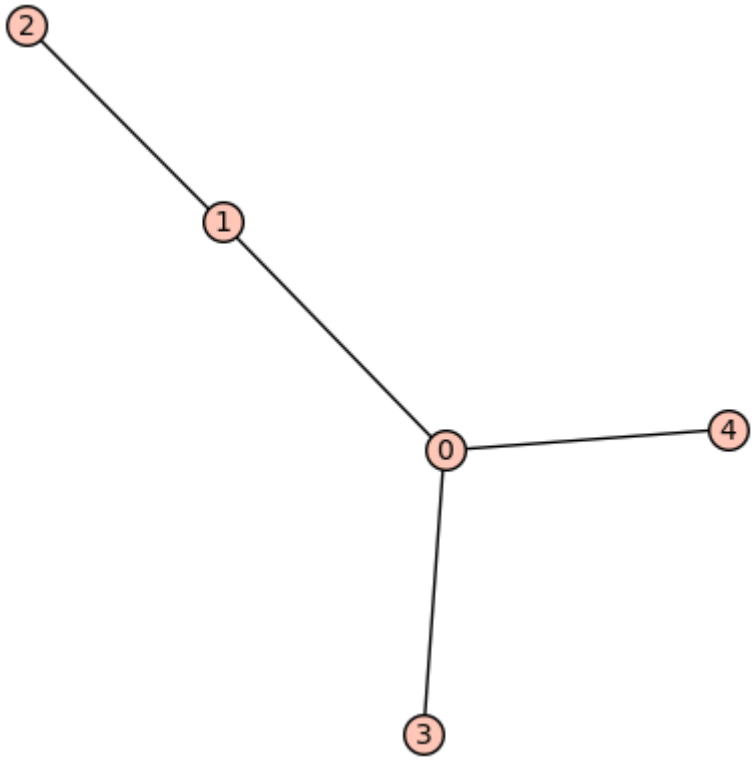
- Ein *Baum* ist ein Graph ohne Kreise. Im Gegensatz zu allgemeinen Graphen können Bäume relativ einfach algorithmisch erzeugt werden, was in Form eines Generators implementiert ist.

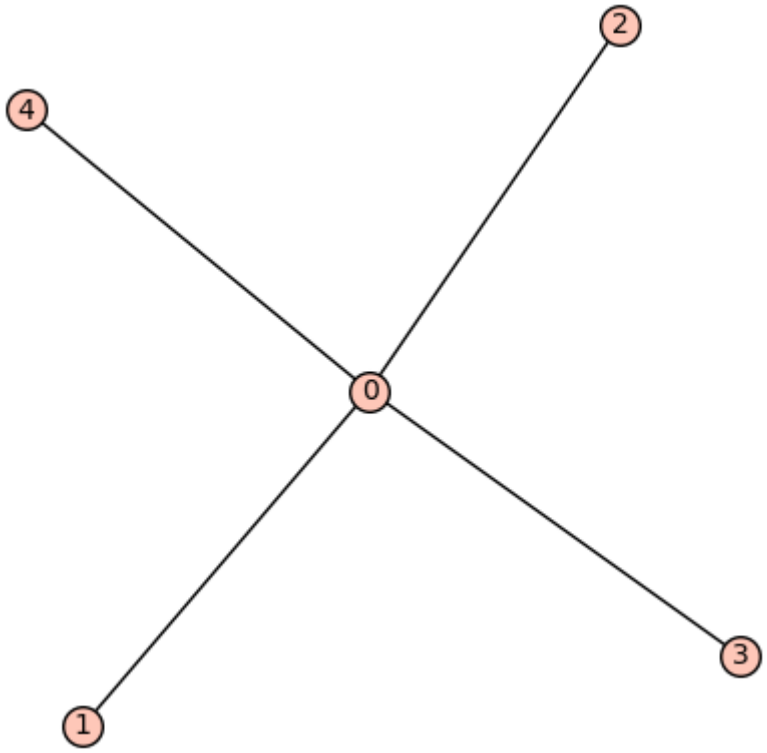
In [86]:
```
t5=graphs.trees(5)
t5
```

Out[86]:
```
<sage.graphs.trees.TreeIterator object at 0x7f324d91e330>
```

In [87]:
```
for t in t5:
    t.show()
```

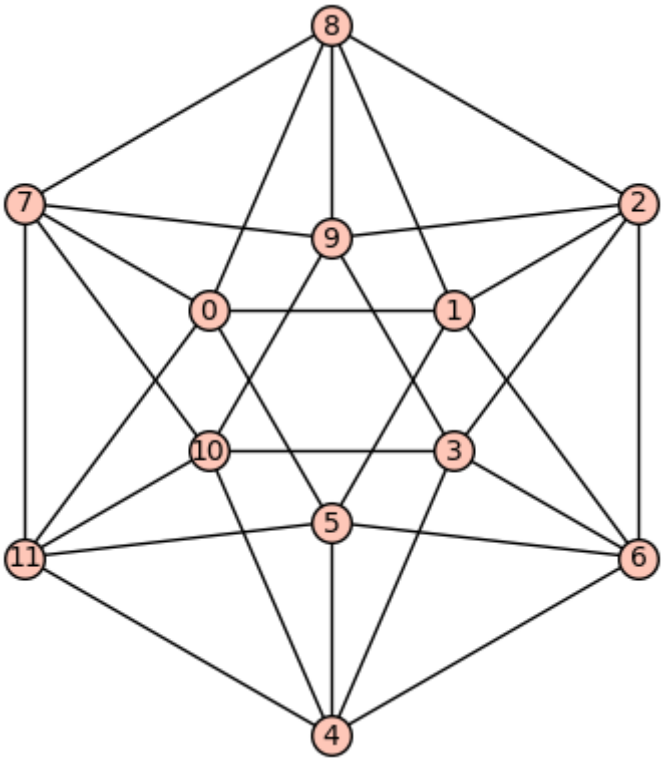




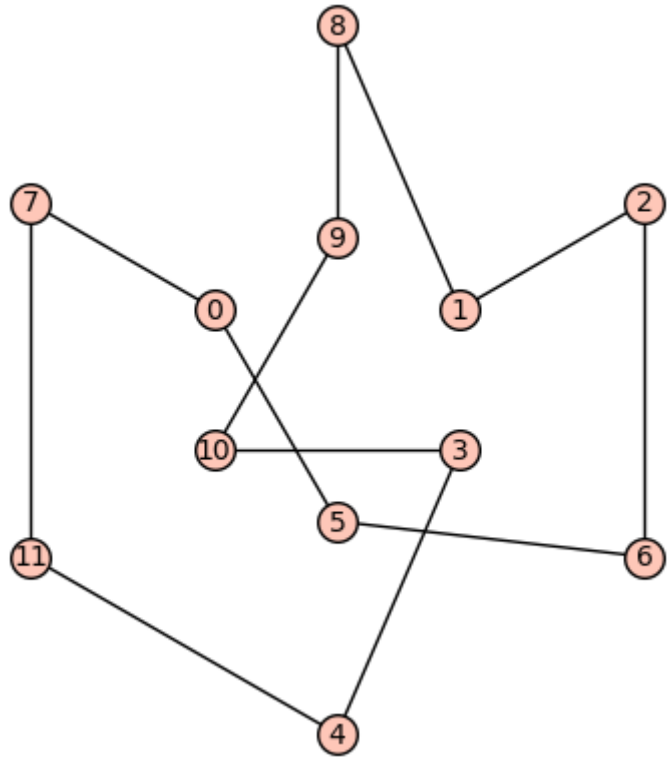


Ein Beispiel eines platonischen Körpers.

In [88]:
```
g20 = graphs.IcosahedralGraph()
g20.show()
```



In [89]:
```
g20.hamiltonian_cycle().show()
```

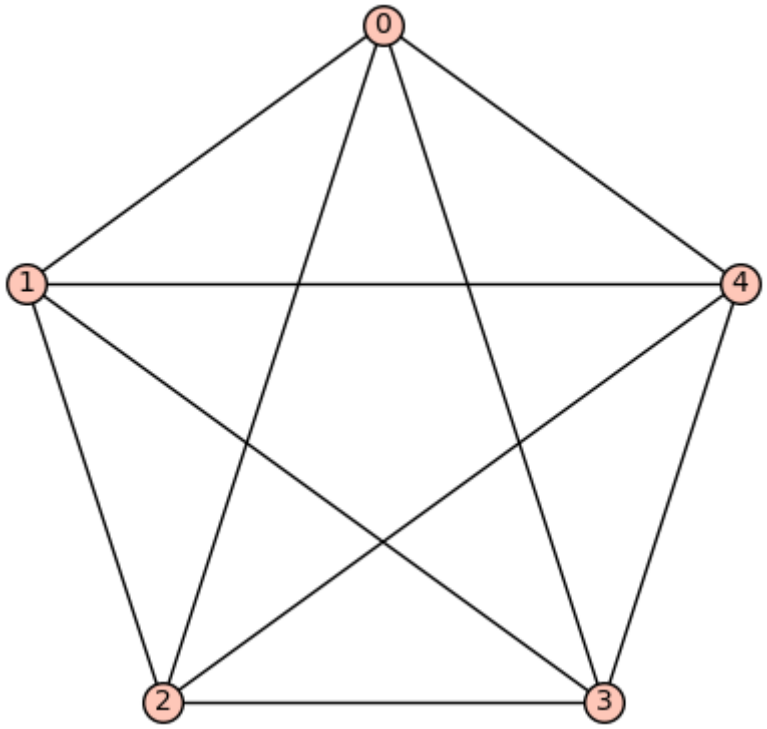


In [90]:
```
g20.eulerian_circuit()
```

Out[90]:
```
False
```

Ein eulerscher Graph muss gerade Knotengrade haben, z.B. der K5

In [91]:
```
k5 = graphs.CompleteGraph(5)
k5.show()
```



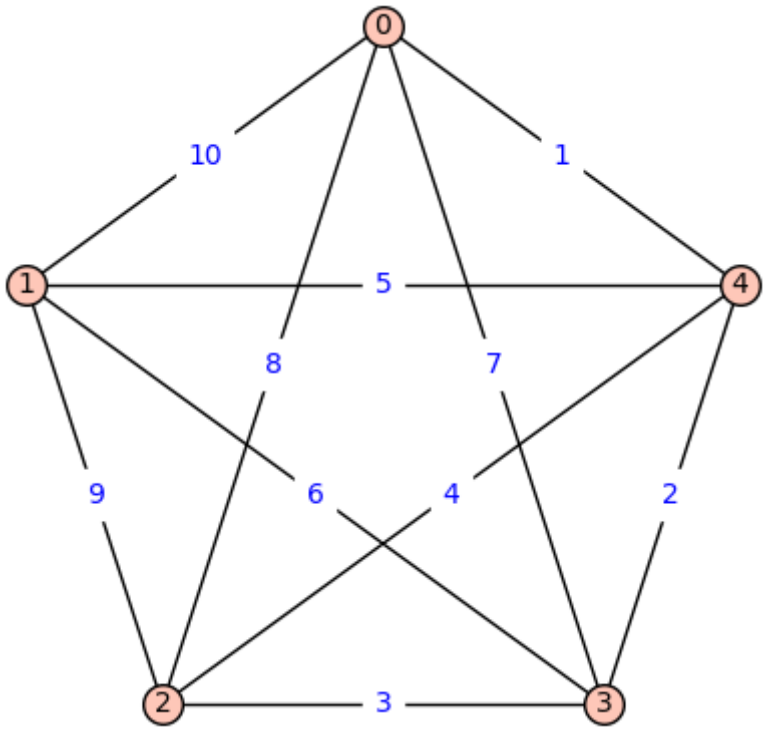
In [92]:
```
k5.eulerian_circuit()
```

Out[92]:
```
[(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)]
```

Um die Reihenfolge sichtbar zu machen, numerieren wir die Kanten anhand dieser Liste durch.

In [93]:
```
for e,i in zip(k5.eulerian_circuit(), (1..)):
    k5.set_edge_label(e[0],e[1],i)
```

In [94]:
```
k5.show(edge_labels=True)
```



In [ ]: