2013-12-20-143617

Mike Zabrocki

June 2, 2014

Contents

```
taylor(1/(1-q)^2,q,0,10)
```

```
11*q^10 + 10*q^9 + 9*q^8 + 8*q^7 + 7*q^6 + 6*q^5 + 5*q^4 + 4*q^3 + 3*q^2 + 2*q + 1
```

```
var('z')
taylor(1/(1-z)^4,z,0,10)
```

```
z
286*z^10 + 220*z^9 + 165*z^8 + 120*z^7 + 84*z^6 + 56*z^5 + 35*z^4 + 20*z^3 + 10*z^2 + 4*z
+ 1
z
```

```
# here is a simple if-then-elif-else
x=2
if x==1:
    print "one!"
elif x==2: # elif is short for "else if"
    print "two!"
else:
    print "not one or two!"
```

```
two!
```

```
# Note: hit return alone to move to the next line
# hit shift and return to evaluate an expression

# to write a function use the notation def functionname( variables ):
# this is an example of a function which converts positive integers into \
   strings
def number_to_word( x ):
    names = ["zero","one","two","three","four","five","six","seven","\
       eight","nine"]
    if x in range(10):
        return names[x] # lists start at position 0!!!!
    elif x<20:
        #do something else
        #should handle numbers 10 through 19
    elif x>=20 and x<100:
        #do somethign else
```

```
        # handle numbers 20 through 99 = "tenword" or "tenword"+" 1 \
            throuth 9 word"
    else:
        return "not zero through one hundred!"
```

```
# Exercise write the function that returns a string for all numbers < 100
#hint add strings and they concatenate
"string 1"+" string 2"
```

```
'string 1 string 2'
```

```
""
number_to_word(0)
```

```
'zero'
```

```
number_to_word(2)
```

```
'two'
```

```
number_to_word(3)
```

```
'three'
```

```
for i in range(0,10):
    number_to_word(i)
```

```
'zero'
'one'
'two'
'three'
'four'
'five'
'not zero through five!'
'not zero through five!'
'not zero through five!'
'not zero through five!'
```

```
range(0,10)
```

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

```
names = ["one","two","three","four","five"]
names[1]
```

```
'two'
```

```
names[0]
```

```
'one'
```

```
names[-1]
```

```
'five'
```

```
# type G = Dih<tab> and then fill in (5)
G = DihedralGroup(5)
```

```
G
```

```
Dihedral group of order 10 as a permutation group
```

```
# once sage knows G is a group, can type G.char<tab>
# and see a list of functions that beggin with "char"
G.character_table()
```

```
[                    1                    1                    1
1]
[                    1                   -1                    1
1]
[                    2                    0      zeta5^3 + zeta5^2 -zeta5^3 - zeta5^2 -
1]
[                    2                    0 -zeta5^3 - zeta5^2 - 1      zeta5^3 +
zeta5^2]
```

```
G.list()
```

```
[(), (2,5)(3,4), (1,2)(3,5), (1,2,3,4,5), (1,3)(4,5), (1,3,5,2,4), (1,4)(2,3),
(1,4,2,5,3), (1,5,4,3,2), (1,5)(2,4)]
```

```
# to list the conjugacy classes of G, start with G as a full set of \
   elements
# and remove the conjugacy class as we calculate it
# step 1: start with set of G
# step 2: take first element in that set and compute conjugacy class
# step 3: subtract off that conjugacy class ... repeat until no elements \
   left
G = SymmetricGroup(4)
Gset = Set(G.list())
while len(Gset)>0:
    g = Gset[0] #let g be the first element of Gset
    CC = Set([h*g*h^(-1) for h in G]) # this is the conjugacy class of g
    Gset = Gset.difference(CC)
    print CC
```

```
{(1,3,2,4), (1,4,3,2), (1,3,4,2), (1,2,3,4), (1,2,4,3), (1,4,2,3)}
{(3,4), (1,2), (2,3), (1,4), (2,4), (1,3)}
{(1,2,3), (1,3,4), (2,3,4), (2,4,3), (1,4,3), (1,2,4), (1,3,2), (1,4,2)}
{(1,4)(2,3), (1,3)(2,4), (1,2)(3,4)}
{()}
```

```
def my_conjugacy_classes( G ):
    Gset = Set(G.list())
    out = []
    while len(Gset)>0:
        g = Gset[0] #let g be the first element of Gset
        CC = Set([h*g*h^(-1) for h in G]) # this is the conjugacy class \
           of g
        Gset = Gset.difference(CC)
        out.append(CC)
    return out
```

```
my_conjugacy_classes(DihedralGroup(4))
```

```
[{(1,4,3,2), (1,2,3,4)}, {(1,3), (2,4)}, {(1,4)(2,3), (1,2)(3,4)}, {()}, {(1,3)(2,4)}]
```

```
my_conjugacy_classes(SymmetricGroup(3))
```

```
[{(1,3), (1,2), (2,3)}, {()}, {(1,2,3), (1,3,2)}]
```

```
my_conjugacy_classes(DihedralGroup(7))
```

```
[{(1,5,2,6,3,7,4), (1,4,7,3,6,2,5)}, {(1,2)(3,7)(4,6), (2,7)(3,6)(4,5), (1,4)(2,3)(5,7),
(1,6)(2,5)(3,4), (1,3)(4,7)(5,6), (1,7)(2,6)(3,5), (1,5)(2,4)(6,7)}, {(1,6,4,2,7,5,3),
(1,3,5,7,2,4,6)}, {(1,7,6,5,4,3,2), (1,2,3,4,5,6,7)}, {()}]
```

```
G.conjugacy_classes()
```

```
[Conjugacy class of () in Symmetric group of order 4! as a permutation group, Conjugacy
class of (1,2) in Symmetric group of order 4! as a permutation group, Conjugacy class of
(1,2)(3,4) in Symmetric group of order 4! as a permutation group, Conjugacy class of
(1,2,3) in Symmetric group of order 4! as a permutation group, Conjugacy class of
(1,2,3,4) in Symmetric group of order 4! as a permutation group]
```

```
SS = Set([1,2,3])
```

```
SS.difference([1])
```

```
{2, 3}
```