「Python による
2 次元・3 次元
グラフィックス」

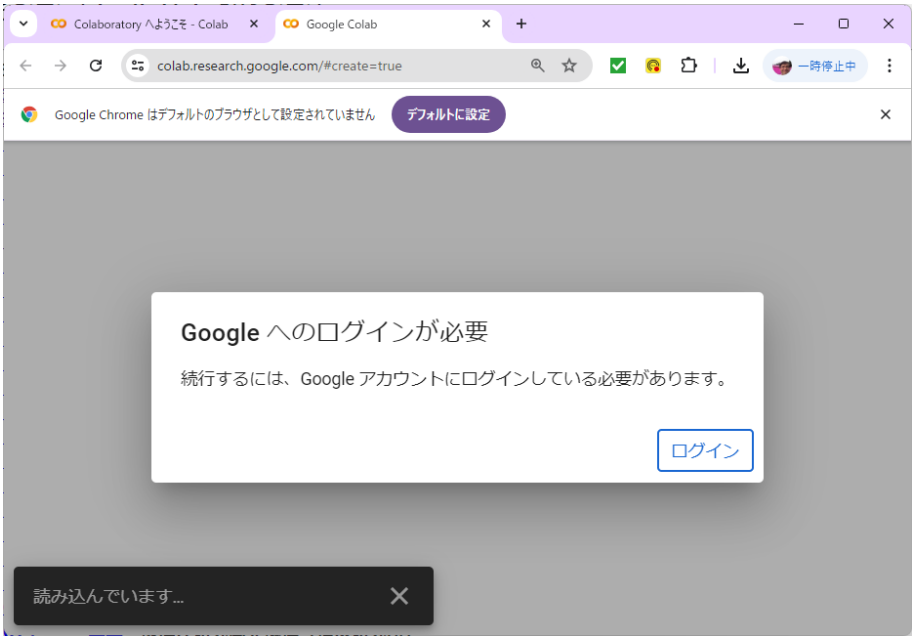
中央情報専門学校

担当：井門俊治（いどしゅんじ）

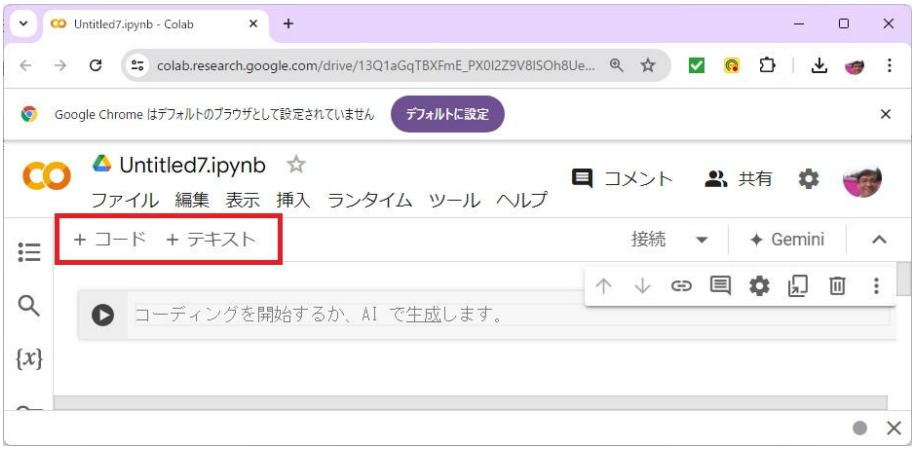
第 1 章 Google Colab の利用と Python プログラミング

Google Colab は Python3 相当のプログラミングを行う Web ベースの環境である。インストールが不要であるため、教育用の PC 教室などでの利用に便利である。

（1）Colab 検索すると、 Colab の Web ページが開く。プログラムの作成、実行などにおいては、Googe のアカウントへのログインが要求される。

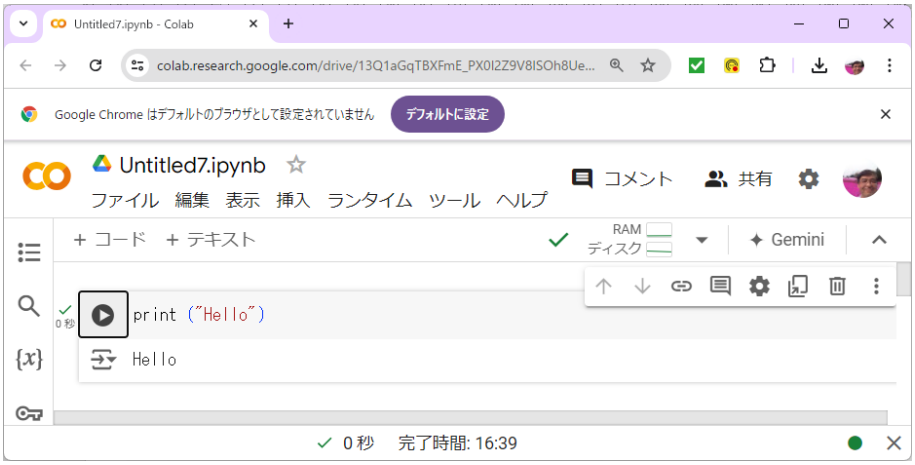


（2）まずあたらしく「ノートブック」を開く。



（3）ノートブックの画面では、記述する部分（セル）には、プログラムを記述する「コードセル」と説明を記述する「テキストセル」が表示される。

(4)セルにプログラムを記述し、矢印を押すと実行される。



Python はインタプリタ言語であり、1 行単位で解釈され、実行される。エラーが発生した場合は、その行で「エラーメッセージ」を出して、停止する。

(5)プログラミング

#1 まず、print 文を使う。文字が表示される。日本語も表示できる。

```
print ("Hello")
```

```
Hello
```

#2 繰り返しでは、for や while を用いる。繰り返しを行う部分は、「段落下げ」(インデント)で示す。インデントの大きさは自由であるが、繰り返しの記述の部分ではインデントの大きさはそろえておく必要がある。

```
print ('Start')

for kk in range(10):  # 「:」が繰り返しを示す。
    print (kk)        # インデントが重要

print ("END")
```

```
kk = 0
while kk < 5:
    kk += 1
    print (f'ひつじが {kk}匹')   # 「f」は表示のフォーマットを示す。
```

```
ひつじが 1匹
ひつじが 2匹
ひつじが 3匹
ひつじが 4匹
ひつじが 5匹
```

#3　分岐の場合も、if 文、else 文の範囲は「:」とインデントで示す。

```
# ディクショナリ（辞書）で、教科の点数を与えている。

score = {'国語':70, '英語':60, '社会':50, '数学':80}

for mykey in score.keys():
    if score[mykey]<60:
        print (f'{mykey}は頑張りましょう')
    else:
        print (f'{mykey}は合格です')
```

```
国語は合格です
英語は合格です
社会は頑張りましょう
数学は合格です
```

#4　関数の活用

まとまった処理には、関数を定義することができる。

```
# 円の面積を求めるプログラム

import math

def circle1(r):
    s = r*r*math.pi
    return s

r = input ('円の半径を入力してください　　')
print (r)
rr = float(r)
s = circle1(rr)
print (s)
```

```
# 関数の利用　階乗(Factorial)の計算

def factorial_cal (x,y):
  x=x*y
  return x

f = 1
for i in range(1,11,1):
  f = factorial_cal(f,i)
  print (i,f)

print ('¥n')
print ('1から10までの階乗は = ',f)
```

Python 自身は簡単なプログラミング言語だが、外部ライブラリなどを活用することで、広く応用することができる。

組み込み関数	print	標準出力に出力
	input	標準入力から 1 行読み込む
	dict	ディクショナリを作成
	len	要素数を返す
標準ライブラリ	math	数学計算
	random	乱数
	datetime	日付と時間
代表的な外部ライブラリ	matplotlib	データの可視化
	pandas	データ解析
	numpy	ベクトル・行列計算
	scipy	科学技術計算
	sympy	代数計算・数式処理
	Scikit-learn	機械学習

（6）Pyhton プログラミングにおけるグラフィック機能

2次元のグラフィック機能としては、上記の外部ライブラリの

matplotlib

が主要なものである。また、ほかの言語でも実現されている「タートル」（カメのアイコンが線画を描く）も Python 用に利用できる。

Turtle

4 年ほど前には、Turtle の機能不足を補う形で、

TurtlePlus

が使えるようになった。今回は、この 2 つのモジュールを共に紹介する。

3 次元グラフィックスとしては、matplotlib の 3 次元拡張版が使える。

第2章　Turtle と TurtlePlus

(1) Turtle による 2 次元グラフィックス

Turtle においては、「カメ(Turtle)」のアイコンが、カンバス上を移動する。penup の状態の場合は、線画を描かないが、pendown の場合は、移動の軌道上に線画を描く。

まず簡単な場合として、五角形を描いてみる。

Turtle モジュールは Colab の中では標準装備されていないため、下記の命令でインストール、取り込み(import)する必要がある。

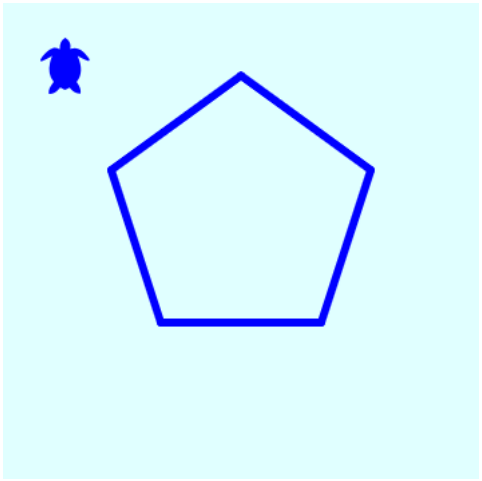
```
!pip3 install ColabTurtle
from ColabTurtle.Turtle import *
```

この操作は、セッション開始時に一度だけ実行すればよい。(ただし、以下のプログラムでは、プログラム実行時に毎回呼び出している。Colab は 2 回目以降は「もうすでに用意済だよ」とメッセージを送ってくるが、エラーではない。

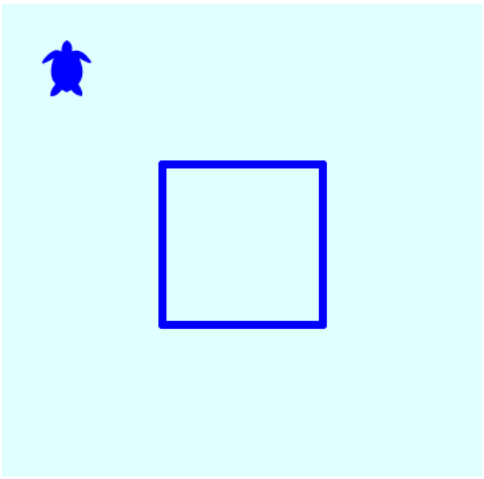
このプログラムにおいては、カメの動きを目視するために移動速度(speed)は、あえて1にしている。

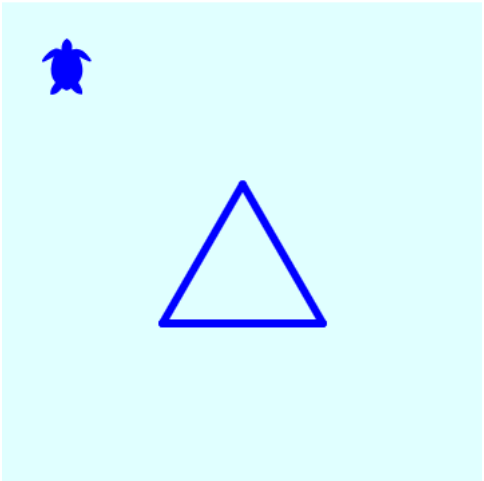
```
# turtle graphics (draw of a pentagon)

initializeTurtle(initial_window_size=(300, 300), initial_speed=1)
color('blue')
bgcolor('lightcyan')

shape('turtle')

penup()
width(5)
setpos(100,200)
setheading(0)
pendown()
n=5
for i in range(n):
    forward(100)
    left(360/n)

penup()
setpos (40,40)
setheading(-90)
```

出力結果を下記に示す。



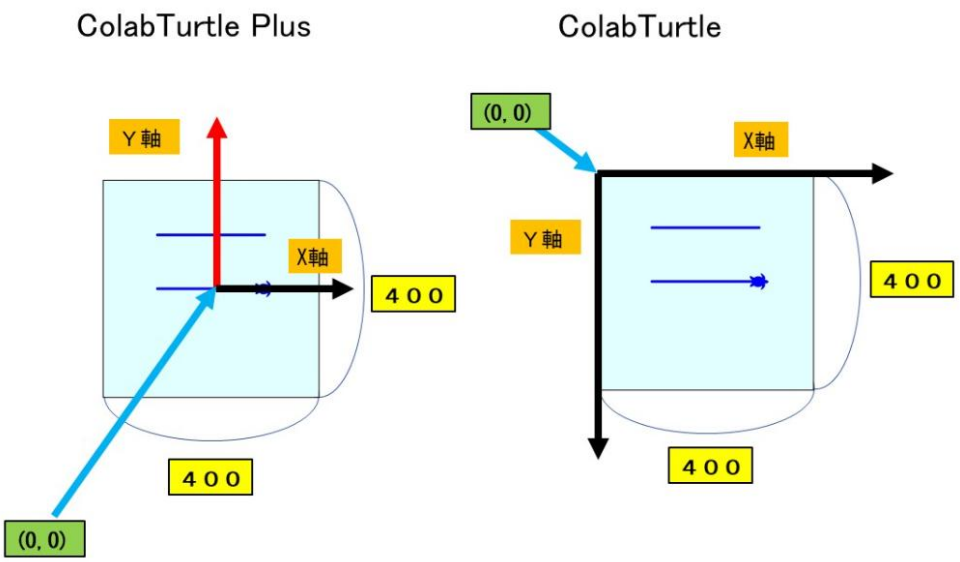
また、n=4 とすると、四角形を描き、n=3 とすると、三角形を描く。





（2）TurtlePlus による 2 次元グラフィックス

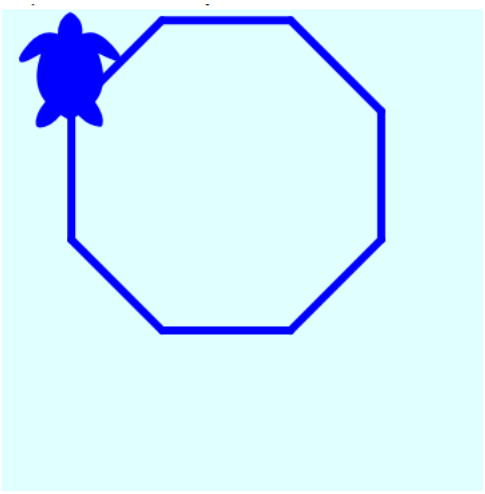
Turtle と TurtlePlus では、描画機能の違いのほかに、原点、座標軸の方向が異なる。



比較のために、TurtlePlus での五角形の描画プログラムを示す。

```
!pip3 install ColabTurtlePlus
from ColabTurtlePlus.Turtle import *

clearscreen()
setup(300,300)
speed(1)              # カメの動きを目視するために speed を1にしている
color('blue')
bgcolor('lightcyan')

shape('turtle')
shapesize(1)       # カメの大きさを変えることができる

penup()
width(5)
setpos(-50,-50)     # Turtle の setpos(100,200)と同じ位置になる
setheading(0)
pendown()
n=5
for i in range(n):
      forward(100)
      left(360/n)

penup()
setpos (-110,110)     # Turtle の setpos(40,40)と同じ位置になる
setheading(90)
```

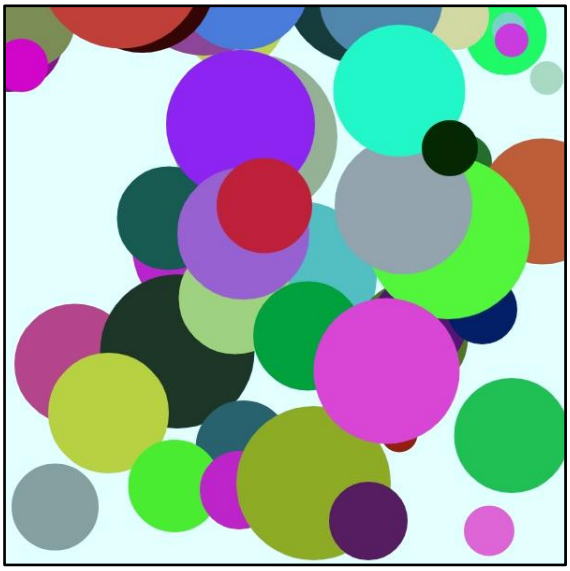
今度は、n=8 として八角形を描いてみる。ただし、先の五角形のプログラムのままでは、300x300 の領域の外にはみ出すため、八角形の辺の長さを 100 から 80 に縮めている。
さらに speed(3) および shapesize(2)の変更も加えている。



```
!pip3 install ColabTurtlePlus
from ColabTurtlePlus.Turtle import *

clearscreen()
setup(300,300)
speed(3)              # カメの動きを目視するために speed を1にしている
color('blue')
bgcolor('lightcyan')

shape('turtle')
shapesize(2)       # カメの大きさを変えることができる

penup()
width(5)
setpos(-50,-50)     # Turtle の setpos(100,200)と同じ位置になる
setheading(0)
pendown()
n=8
for i in range(n):
      forward(80)
      left(360/n)

penup()
setpos (-110,110)     # Turtle の setpos(40,40)と同じ位置になる
setheading(90)
```

(3)乱数による水玉模様

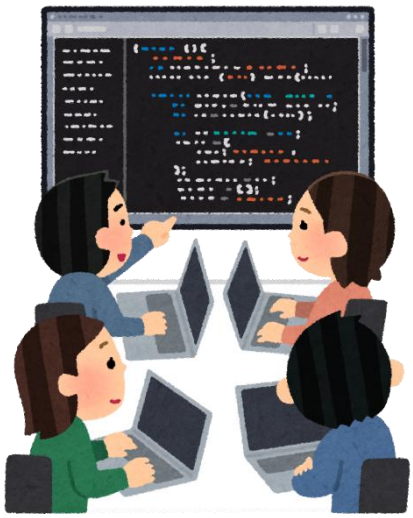
TutlePlus では、塗りつぶし機能や円を描く関数 circle が入ったので、乱数により、色、位置、大きさを決めて水玉模様を描くことができる。
プログラムの実行例を下記に示す。この結果は、乱数により決めているので、実行のたびに違う結果が得られる。



プログラムは下記に示す。

```
!pip3 install ColabTurtlePlus
from ColabTurtlePlus.Turtle import *

import random

clearscreen()
setup(400,400)
speed(10)                  # speed を 10 にして速く描画する
color('blue')
bgcolor('lightcyan')

shape('turtle')
shapesize(2)

showborder()
color("red", "yellow")

pensize(2)

for n in range(60):
    # 乱数により色を決める
```

```
        r1=int(250*random.random())
        g1=int(250*random.random())
        b1=int(250*random.random())
        color ((r1,g1,b1))
        fillcolor((r1,g1,b1))
        penup()

        # 乱数により位置、大きさを決める
        x1=int(-200+400*random.random())
        y1=int(-200+400*random.random())
        radius1=10+int(50*random.random())

        #print(n, r1, g1, b1, x1, y1, radius1)
        setpos(x1, y1)

        pendown()
        begin_fill()
        circle(radius1)
        end_fill()

hideturtle()        # カメを消す
```



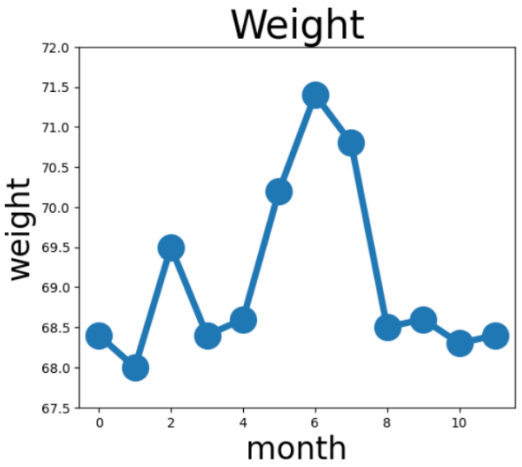
第3章　グラフ表示のためのライブラリ　matplotlib

先に挙げた外部ライブラリとして、グラフ表示を行うためのライブラリ matplotlib が用意されている。これを用いたグラフ表示の例を図示する。

(1) 折れ線グラフを作成する。

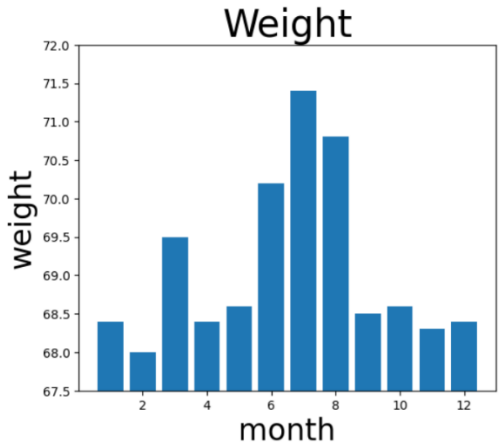
プログラムは下記に示す。y 方向の値は、リストであたえている。

また、ここでは、「マーカー」を用いている。



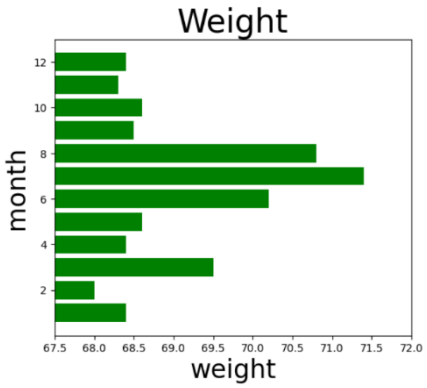

```
# 折れ線グラフ

import matplotlib.pyplot as plt
weight = [68.4, 68.0, 69.5, 68.4, 68.6, 70.2, 71.4, 70.8,
          68.5, 68.6, 68.3, 68.4]

plt.figure(figsize=(6, 5))
plt.title("Weight", fontsize=30)
plt.xlabel("month", fontsize=24)
plt.ylabel("weight", fontsize=24)
plt.ylim(67.5, 72.0)
plt.plot (weight,marker="o", ms=20, lw=5)
# マーカーのサイズ、線の太さを設定

plt.show()
```

(2) 棒グラフ(bar)を作成する。




```
# 棒グラフ

import matplotlib.pyplot as plt
weight = [68.4, 68.0, 69.5, 68.4, 68.6, 70.2, 71.4, 70.8,
          68.5, 68.6, 68.3, 68.4]
month = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11, 12]

plt.figure(figsize=(6, 5))
plt.title("Weight", fontsize=30)
plt.xlabel("month", fontsize=24)
plt.ylabel("weight", fontsize=24)
plt.ylim(67.5, 72.0)

plt.bar(month,weight)
plt.show()
```

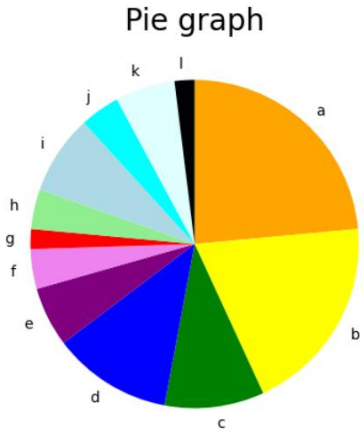
(3) 横棒グラフを作成する。



bar が barh に代わっている。

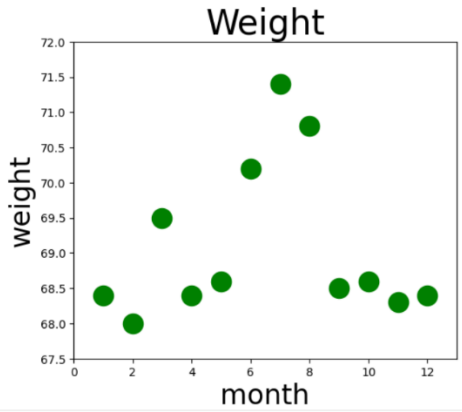
```
# 横棒グラフを作成する

import matplotlib.pyplot as plt
weight = [68.4, 68.0, 69.5, 68.4, 68.6, 70.2, 71.4, 70.8,
          68.5, 68.6, 68.3, 68.4]
month = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11, 12]
plt.figure(figsize=(6, 5))
plt.title("Weight", fontsize=30)
plt.ylabel("month", fontsize=24)
plt.xlabel("weight", fontsize=24)

plt.xlim(67.5, 72.0)
# 変数軸(ここではx軸)の値を設定

plt.barh(month,weight, color='green')
# 横棒グラフ

plt.show()
```

(4) 円グラフ




```
# 円グラフ

import matplotlib.pyplot as plt
value = [12, 10, 5, 6, 3, 2, 1, 2, 4, 2, 3, 1]
color1 = ['orange', 'yellow', 'green', 'blue', 'purple', 'violet', 'red',
          'lightgreen', 'lightblue', 'cyan', 'lightcyan', 'black']
label1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l']
print (value)

plt.figure(figsize=(8, 5))
plt.title("Pie graph", fontsize=20)
plt.pie (value, colors=color1, startangle=90,
counterclock=False, labels=label1)

plt.show()
```

(5) 散布図の作成



散布図においては、マーカーを用いている。ここでは、デフォールトの

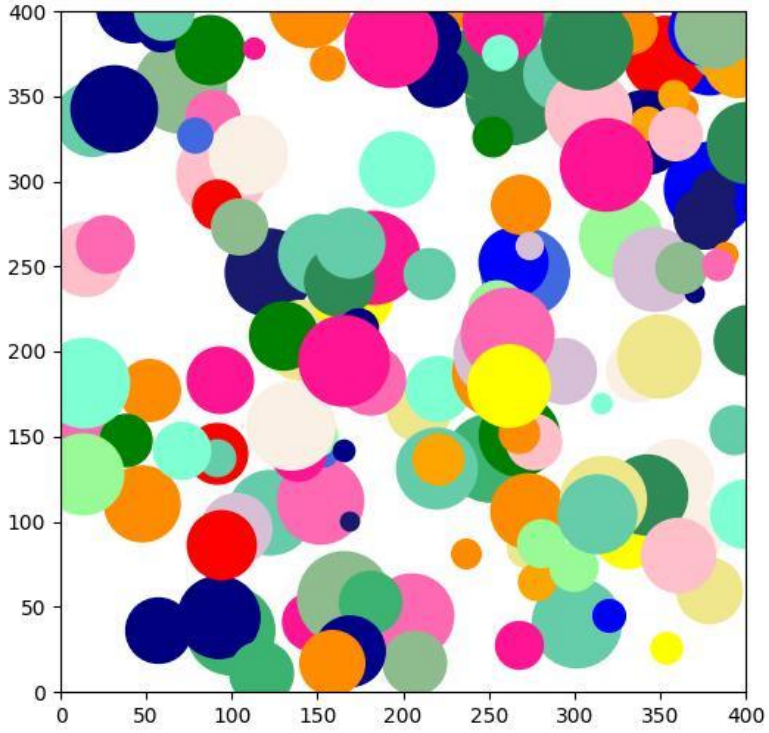
marker="o"

が使われている。すなわち、「塗りつぶした円」である。

```
# 散布図

import matplotlib.pyplot as plt
weight = [68.4, 68.0, 69.5, 68.4, 68.6, 70.2, 71.4, 70.8,
          68.5, 68.6, 68.3, 68.4]
month = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11, 12]

plt.figure(figsize=(6, 5))
plt.title("Weight", fontsize=30)
plt.xlabel("month", fontsize=24)
plt.ylabel("weight", fontsize=24)

plt.ylim(67.5, 72.0)              # 縦軸の値を設定
plt.xlim(0,13)

plt.scatter(month,weight, color='green', s=300)

plt.show()
```

(6) マーカーを用いたグラフィックス 「水玉模様」

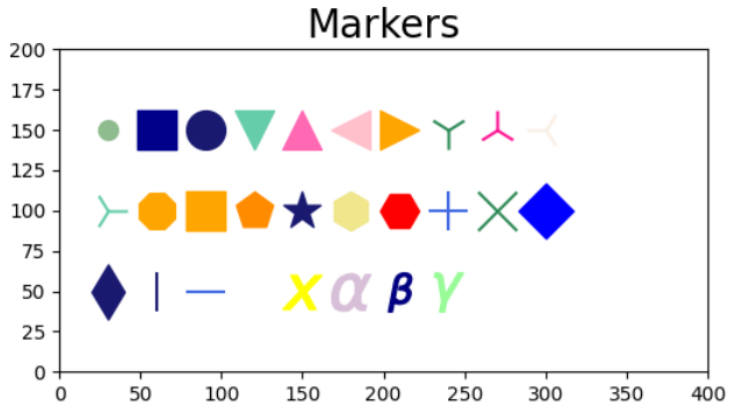
この「塗りつぶした円」を使うことで、先の TurtlePlus で実現した「乱数による水玉模様」のグラフィックスを描くことができる。



この時のプログラムを下記に示す。

```
# 散布図　乱数による水玉

import matplotlib.pyplot as plt

# 色の名前の設定
# 下記が参考になります。
# https://www.colordic.org/

color_star = ['blue', 'green', 'red', 'pink', 'yellow',
                    'royalblue', 'seagreen', 'khaki', 'deeppink', 'midnightblue',
                    'mediumseagreen', 'hotpink',    'navy', 'mediumaquamarine',
                    'darkblue', 'darkseagreen','orange', 'aquamarine',
                    'palegreen', 'darkorange', 'thistle', 'linen' ]

plt.figure(figsize=(6, 6))
plt.xlim(0,400)
plt.ylim(0,400)
import random
for kk in range(50):
    x0=10+400*random.random()
    y0=10+400*random.random()
    length=5+200*random.random()
    icolor =int(22*random.random())
    color1 = color_star[icolor]
    plt.scatter(x0,y0,s=length*10,color=color1)

plt.show()
```

(7) マーカーの種類は多くある。
その一部を下記に示す。



この 28 個のマーカーの一覧を表示するプログラムは下記のようなものである。

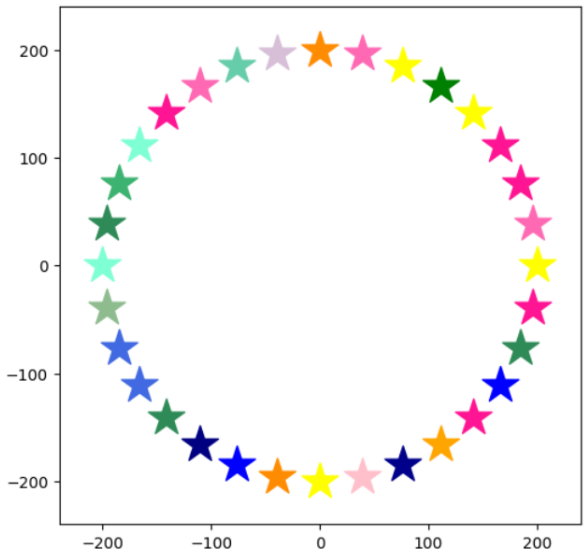
```
# species od markers

import matplotlib.pyplot as plt

# 色の名前の設定
# 下記が参考にあります。
# https://www.colordic.org/

color_star = ['blue', 'green', 'red', 'pink', 'yellow',
                    'royalblue', 'seagreen', 'khaki', 'deeppink', 'midnightblue',
                    'mediumseagreen', 'hotpink',   'navy', 'mediumaquamarine',
                    'darkblue', 'darkseagreen','orange', 'aquamarine',
                    'palegreen', 'darkorange', 'thistle', 'linen' ]
markerss = [".", ",", "o", "v", "^", "<", ">", "1", "2", "3",
                "4", "8", "s", "p", "*", "h", "H", "+", "x", "D",
                "d", "|", "_", "None", "$x$",
                "$¥¥alpha$", "$¥¥beta$", "$¥¥gamma$"]

plt.figure(figsize=(6, 3))
plt.title("Markers", fontsize=20)
plt.xlim(0,400)
plt.ylim(0,200)

import random
for kk in range(28):
    mm = kk % 10   # 割り算のあまり(剰余)
    x0=30*(mm+1)
    y0=200-50*(int(kk/10)+1)
    length=40
    icolor =int(22*random.random())
    color1 = color_star[icolor]
    marker1 = markerss[kk]
    plt.scatter(x0,y0,s=length*10,color=color1,marker=marker1)

plt.show()
```

(8) マーカーによるグラフィックス （リング）

ここでは、マーカーとして、

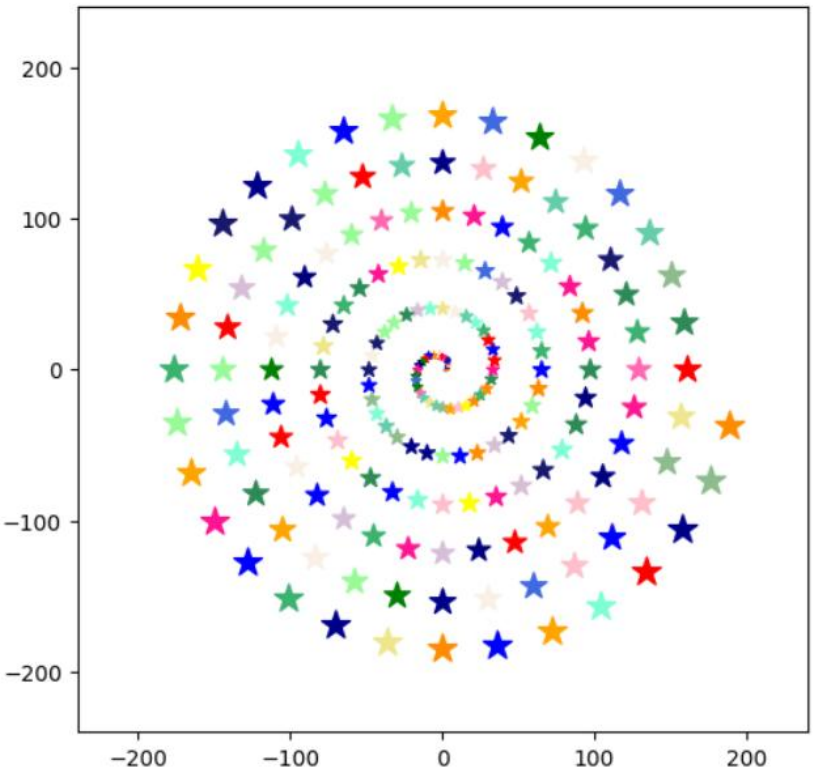
marker="*"

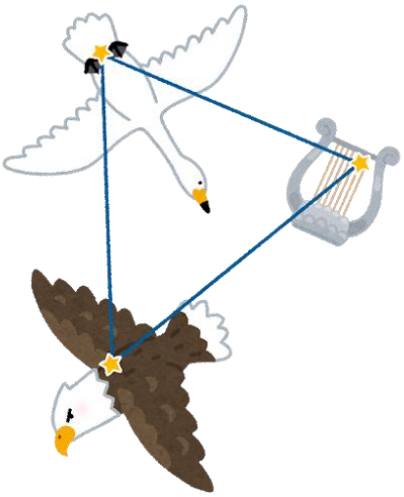
すなわち、「星型」を用いている。円周上にマーカーを並べているが、色は乱数により決定している。




```
# ring of stars

import matplotlib.pyplot as plt

color_star = ['blue', 'green', 'red', 'pink', 'yellow',
                    'royalblue', 'seagreen', 'khaki', 'deeppink', 'midnightblue',
                    'mediumseagreen', 'hotpink',    'navy', 'mediumaquamarine',
                    'darkblue', 'darkseagreen','orange', 'aquamarine',
                    'palegreen', 'darkorange', 'thistle', 'linen' ]

plt.figure(figsize=(6, 6))
plt.xlim(−240,240)
plt.ylim(−240,240)
import random
import math

nn = 32
da = 2* math.pi / nn
for kk in range(nn):
    ang1 = da * kk
    x0=200*math.cos(ang1)
    y0=200*math.sin(ang1)
    length=600
    icolor =int(22*random.random())
    color1 = color_star[icolor]
    plt.scatter(x0,y0,s=length,color=color1,marker="*")

plt.show()
```

(9) マーカーによるグラフィックス(らせん)





```
# spiral of stars

import matplotlib.pyplot as plt

# 色の名前の設定
# 下記が参考にあります。
# https://www.colordic.org/

color_star = ['blue', 'green', 'red', 'pink', 'yellow',
                    'royalblue', 'seagreen', 'khaki', 'deeppink', 'midnightblue',
                    'mediumseagreen', 'hotpink',   'navy', 'mediumaquamarine',
                    'darkblue', 'darkseagreen','orange', 'aquamarine',
                    'palegreen', 'darkorange', 'thistle', 'linen' ]

plt.figure(figsize=(6, 6))
plt.xlim(-240,240)
plt.ylim(-240,240)
import random
import math

nn = 32
da = 2* math.pi / nn
dr = 1
rr = 0
for tt in range(6):
  for kk in range(nn):
    ang1 = da * kk
    rr = rr + dr
    x0=rr*math.cos(ang1)
    y0=rr*math.sin(ang1)
    length=rr
    icolor =int(22*random.random())
    color1 = color_star[icolor]
    plt.scatter(x0,y0,s=length,color=color1,marker="*")

plt.show()
```

第4章　plot による線画グラフィックス

（1）　摂動のある円

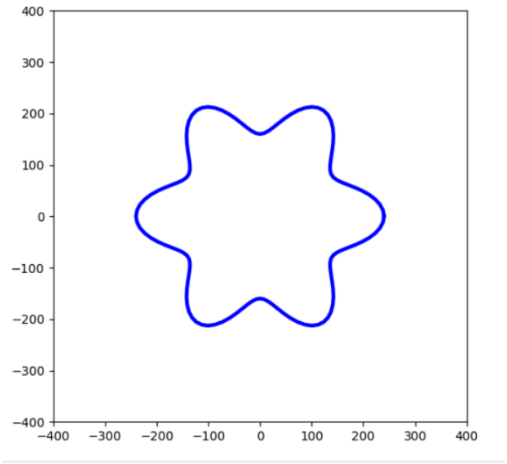
$x = r*\cos(\Theta)$

$y = r*\sin(\Theta)$

$0 \leqq \Theta \leqq 2\pi$

とすると、円が描かれる。

ここで、半径rが変動すると、摂動のある円が描かれる。



右のプログラムにおいて、

半径の変動の割合　dr

変動の周波数の数　m

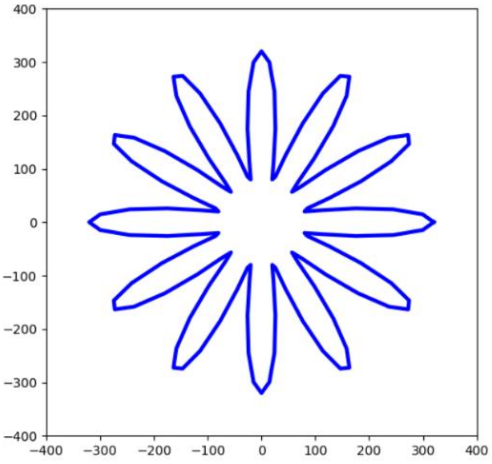
を変えることで、異なった図形を描くことができる。

下図は、

dr=0.6

m=12

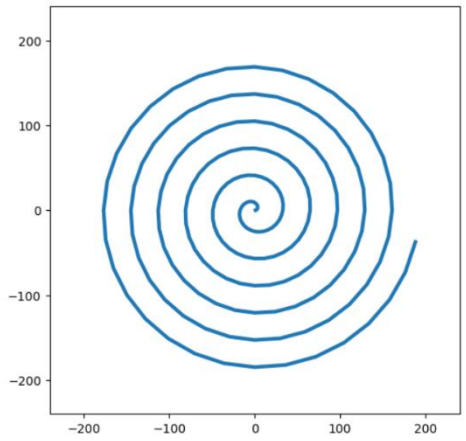
の場合の結果である。




```
# plot graphics

import matplotlib.pyplot as plt

plt.figure(figsize=(6, 6))
plt.xlim(−400,400)
plt.ylim(−400,400)

import random
import math

nn = 128
da = 2* math.pi / nn

xx = [0]*(nn+1)
yy = [0]*(nn+1)

r0 = 200

dr = 0.2
m = 6
for i in range(0,nn+1):
    ang = da * i
    rr =   dr*r0*math.cos(m*ang)
    x2=(r0+rr)*math.cos(ang)
    y2=(r0+rr)*math.sin(ang)
    xx[i] = x2
    yy[i] = y2

plt.plot (xx, yy, linewidth=3, color="blue")
```

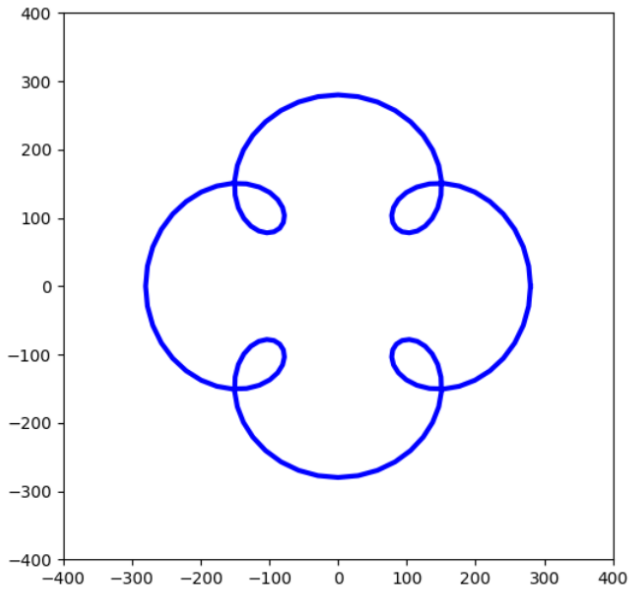
(2)らせん図形

円を描くときに半径rが次第に大きくなると、らせん図形となる。




```
# plotr graphics

import matplotlib.pyplot as plt

plt.figure(figsize=(6, 6))
plt.xlim(-240,240)
plt.ylim(-240,240)
import random
import math

xx = [0]*nn*6
nn = 32
da = 2* math.pi / nn
dr = 1
rr = 0

xx = [0]*nn*6
yy = [0]*nn*6

mm = 0
for tt in range(6):
      for kk in range(nn):
         ang1 = da * kk
         rr = rr + dr
         x0=rr*math.cos(ang1)
         y0=rr*math.sin(ang1)

         xx[mm] = x0
         yy[mm] = y0
         mm = mm+1

plt.plot(xx,yy, linewidth=3)

plt.show()
```

(3)特殊な摂動の円



今度の場合は、やや特殊な摂動を入れている。
具体的な式は、プログラムを参照されたい。

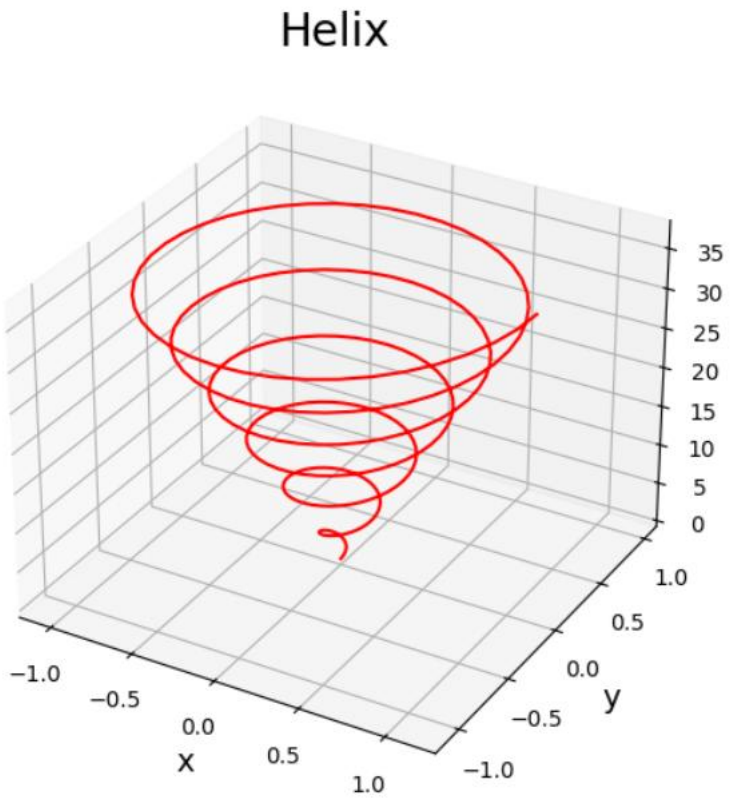
やはり、
摂動の大きさ　dr
摂動の周波数　m
を変えると、描かれる図形の形が変わっていく。
各自試されたい。

```
# plot graphics

import matplotlib.pyplot as plt

plt.figure(figsize=(6, 6))
plt.xlim(−400,400)
plt.ylim(−400,400)

import math
nn = 128
da = 2* math.pi / nn

xx = [0]*(nn+1)
yy = [0]*(nn+1)
r0 = 200

dr = 0.4
m = 5
for kk in range(0,nn+1):
    ang = da*kk
    xx[kk]=r0*math.cos(ang) + dr*r0*math.cos(m*ang)
    yy[kk]=r0*math.sin(ang) + dr*r0*math.sin(m*ang)

plt.plot (xx, yy, linewidth=3, color="blue")
plt.show()
```

第 5 章　3 次元グラフィックス

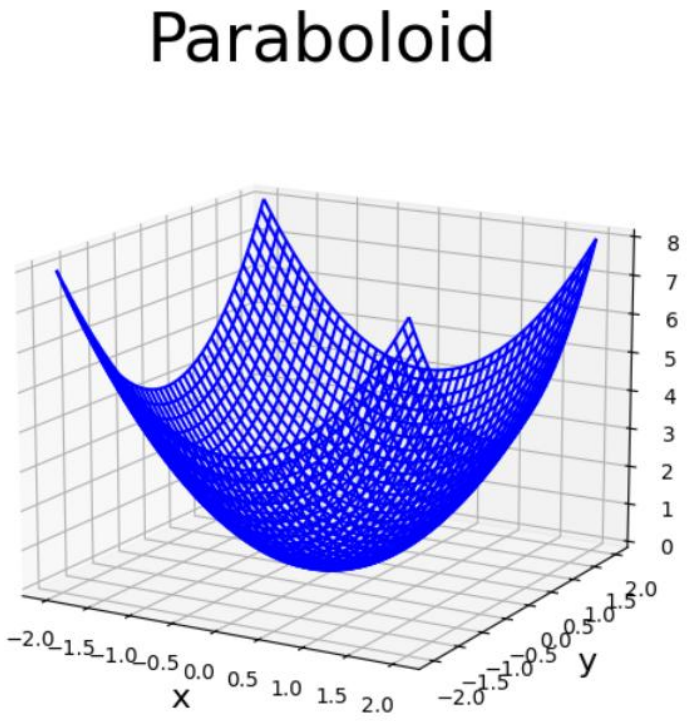
（1）直交座標を用いた 3 次元図形表示

3 次元のらせんを描くプログラムを示す。



```
# PYTHON_MATPLOTLIB_3D_PLOT_02
# 3 次元データの可視化
# https://python.atelierkobato.com/axes3d/ より

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Figure を追加
fig = plt.figure(figsize = (6, 6))

# 3DAxes を追加
ax = fig.add_subplot(111, projection='3d')

# Axes のタイトルを設定
ax.set_title("Helix", size = 20)

# 軸ラベルを設定
ax.set_xlabel("x", size = 14)
ax.set_ylabel("y", size = 14)
ax.set_zlabel("z", size = 14)

# 軸目盛を設定
ax.set_xticks([-1.0, -0.5, 0.0, 0.5, 1.0])
ax.set_yticks([-1.0, -0.5, 0.0, 0.5, 1.0])

# 円周率の定義
pi = np.pi

# パラメータ分割数
n = 256

# パラメータ t を作成
t = np.linspace(0, 12*pi, n)

# らせんの方程式
x = np.cos(t)*t*0.03
y = np.sin(t)*t*0.03
z = t

# 曲線を描画
ax.plot(x, y, z, color = "red")

plt.show()
```

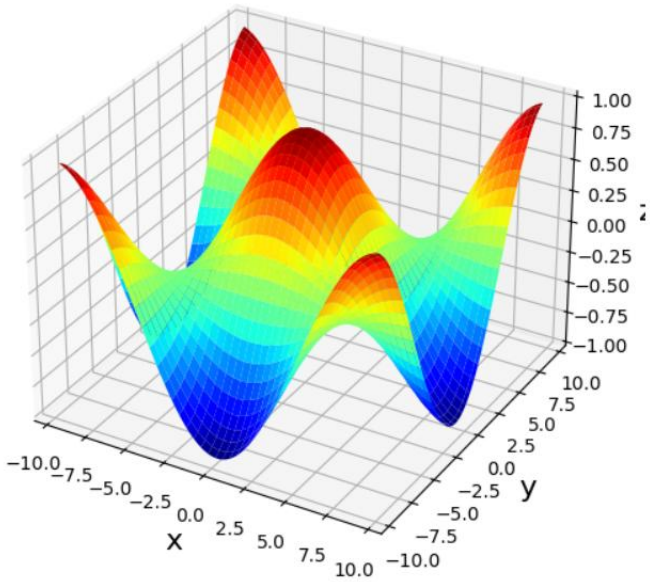
次は、放物面を描画してみる。




```
# PYTHON_MATPLOTLIB_WIREFRAME
#    https://python.atelierkobato.com/surface/

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

# Figure を追加
fig = plt.figure(figsize = (6, 6))

# 3DAxes を追加
ax = fig.add_subplot(111, projection="3d")

# Axes(サブプロット)のタイトルを設定
ax.set_title("Paraboloid", size = 32)
#  軸ラベルを設定
ax.set_xlabel("x", size = 16)
ax.set_ylabel("y", size = 16)
ax.set_zlabel("z", size = 16)
```

```
# (x,y)データを作成
x = np.linspace(-2, 2, 257)
y = np.linspace(-2, 2, 257)

#  格子点の作成
X, Y = np.meshgrid(x, y)

#  高度の計算式
Z = X**2 + Y**2

#  ワイヤーフレームで 3 次元の放物面を描く
ax.plot_wireframe(X, Y, Z, color = "blue")

#  視点を仰角  15°  、方位角  -60°  に設定
#  （デフォールト値は、仰角  30°  、方位角  -60）
ax.view_init(elev=15, azim=-60)

plt.show()
```

今度は、z=cos(x)＊cos(y) を描いてみる。



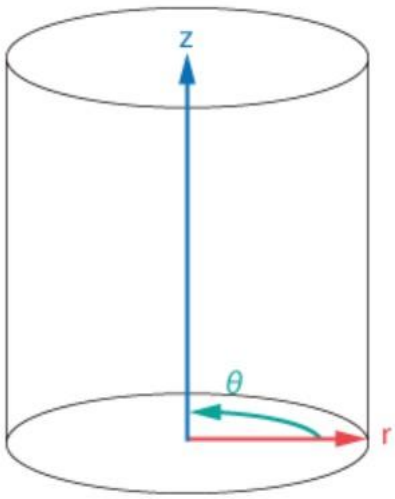

```
# PYTHON_MATPLOTLIB_3D_PLOT_03

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Figure と 3DAxeS
fig = plt.figure(figsize = (6, 6))
ax = fig.add_subplot(111, projection="3d")

#  軸ラベルを設定
ax.set_xlabel("x", size = 16)
ax.set_ylabel("y", size = 16)
ax.set_zlabel("z", size = 16)

#  円周率の定義
pi = np.pi

# (x,y)データを作成
x = np.linspace(-3*pi, 3*pi, 256)
y = np.linspace(-3*pi, 3*pi, 256)
```

```
#  格子点を作成
X, Y = np.meshgrid(x, y)

#  高度の計算式
Z = np.cos(X/pi) * np.cos(Y/pi)

#  曲面を描画
ax.plot_surface(X, Y, Z, cmap = "jet")

#  底面に等高線を描画
#ax.contour(X, Y, Z, colors = "black", offset = -1)

plt.show()
```

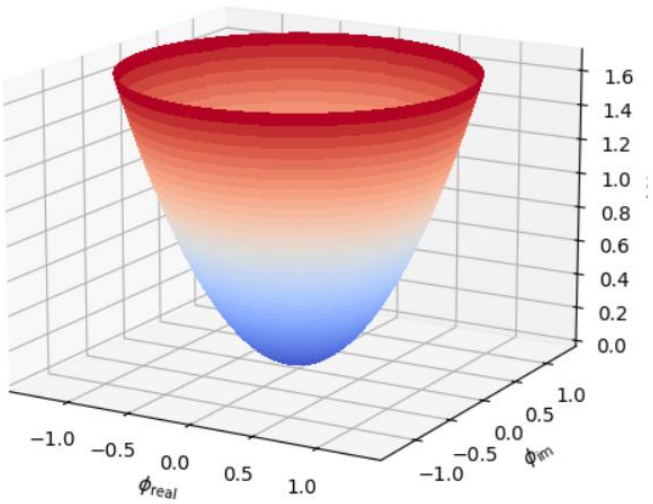
(2)円筒座標を用いた 3 次元図形表示

直交座標系（x, y, z）と円筒座標系（r, θ , z）の間には以下の関係が成り立つ。

$$x = r\cos\theta$$
$$y = r\sin\theta$$
$$z = z$$

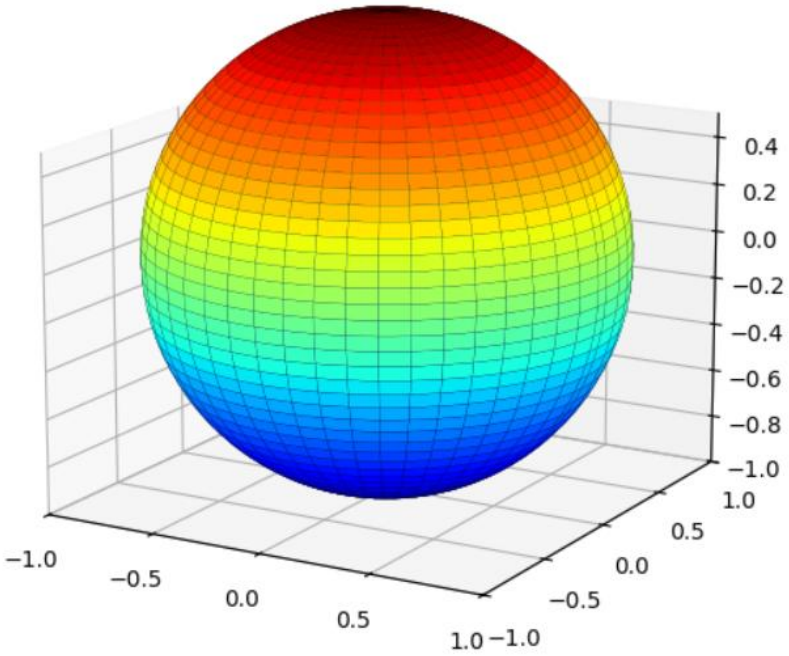


下図は、z=r**2 = (sqrt(x**2+y**2))**2 を描いている。



```
#  PYTHON_MATPLOTLIB_WIREFRAME
#   https://python.atelierkobato.com/surface/

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

# Figure を追加
fig = plt.figure(figsize = (6, 6))

# 3DAxes を追加
ax = fig.add_subplot(111, projection="3d")

# Create the mesh in polar coordinates and compute
corresponding Z.
r = np.linspace(0, 1.3, 50)
p = np.linspace(0, 2*np.pi, 50)
R, P = np.meshgrid(r, p)
Z = R**2

# Express the mesh in the cartesian system.
X, Y = R*np.cos(P), R*np.sin(P)

# Plot the surface.
ax.plot_surface(X, Y, Z, cmap=plt.cm.coolwarm,antialiased=False)
# Tweak the limits and add latex math labels.
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

#  仰角、方位角を設定
ax.view_init(elev=15, azim=-60)

plt.show()
```

(3)球座標を用いた 3 次元図形表示




```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from mpl_toolkits.mplot3d import axes3d

r=1
p = np.linspace(0, 2* np.pi, 100)
t = np.linspace(0, np.pi, 100)
p, t = np.meshgrid(p, t)

x = r* np.sin(t)* np.cos(p)
y = r* np.sin(t)* np.sin(p)
z = r* np.cos(t)

# plotting
fig = plt.figure(figsize=(6,8))
ax = fig.add_subplot( 111 , projection='3d')
```

```
ax.set_xlim(−1,1)
ax.set_ylim(−1,1)
ax.set_zlim(−1,0.5)

ax.set_xticks([−1,−0.5, 0, 0.5, 1])
ax.set_yticks([−1,−0.5, 0, 0.5, 1])
#ax.set_zticks([−1,−0.5, 0, 0.5, 1])

ax.plot_surface(x, y, z, linewidth = .1,
edgecolor='k',cmap='jet', antialiased=True)

# 視点の変更、仰角・方位角の設定
ax.view_init(elev=15, azim=−60)

plt.show()
```

この球の表現において、半径rの値に変動を加えた時の図形を下記に示す。
パラメータの説明は、プログラムの説明時に与える。