〈[파이썬 데이터 시각화 마법] 제1장: 점과 선으로 그리는 이야기, Matplotlib!〉

선생님: (다양한 색상의 물감과 붓, 그리고 그래프가 그려진 종이를 보여주며) 얘들아, 안녕! 우리가 숫자로 세상을 분석하고, 코드로 문제를 해결했지? 오늘은 그 결과를 다른 사람들에게 한눈에 보여주거나, 데이터 속에 숨겨진 이야기를 찾아내는 '그림 마법', 바로 **데이터 시각화**를 배워볼 거야! 파이썬에는 이 마법을 도와주는 **Matplotlib**이라는 아주 유명한 화가가 있단다.

코딩새싹 🌱: 그림 마법이요? 우리가 직접 그래프를 그릴 수 있는 거예요?

선생님: 물론이지! 먼저, Matplotlib 화가를 우리 작업실로 모셔와야겠지? 보통 `plt`라는 애칭으로 부른단다.

```
import matplotlib.pyplot as plt # Matplotlib의 그림 그리기 도구모음을 plt라는 이름으로 불러오자!
import numpy as np # 복잡한 숫자 목록이나 수학 계산을 위해 넘파이 마법사도 함께!
```

〈1. 캔버스에 점 찍기: `plt.scatter()`〉

선생님: 가장 기본적인 그림은 바로 점을 찍는 거야! `plt.scatter(x좌표, y좌표)` 라고 하면, 우리 캔버스에 예쁜 점을 콩! 하고 찍어준단다.

```
# x가 4, y가 2인 위치에 점 하나!
plt.scatter(4, 2)
# x가 3, y가 1인 위치에도 점 하나!
plt.scatter(3, 1)
# plt.show() # Colab이나 Jupyter 환경에서는 자동으로 보여주지만, 다른 곳에선 이 주문이 필요해!
```

선생님: 여러 개의 점을 한 번에 찍을 수도 있어. x좌표들을 리스트로, y좌표들을 리스트로 만들어서 넘겨주면 되지! 점의 크기(s), 색깔(c), 투명도(alpha)도 바꿀 수 있단다.

```
x_points = [1, 4]
y_points = [2, 8]
plt.scatter(x_points, y_points, s=200, c="green", alpha=0.4) # 크기는 200, 색은 초록, 투명도는 0.4!
# plt.show()
```

궁금이 💡: alpha는 투명도군요! 0에 가까울수록 투명해지고, 1에 가까울수록 진해지는 건가요?

선생님: 정확해! 0은 완전 투명, 1은 완전 불투명을 의미하지.

〈2. 점들을 이어라! `plt.plot()`〉

선생님: 점들을 찍었다면, 그 점들을 선으로 스르륵~ 이어볼 수도 있어! `plt.plot(x좌표들, y좌표들)` 마법을 쓰면 된단다.

```
# 아까 찍었던 점들을 선으로 이어볼까?
# plt.plot(x_points, y_points) # 기본 파란색 선으로!
# plt.show()

# 그럼 y = x² + 1 그래프를 그려볼까? x가 -10부터 10까지 변할 때!
x_values = range(-10, 11) # -10, -9, ..., 0, ..., 9, 10
```

```
y_values = list(map(lambda i: i**2 + 1, x_values)) # 각 x에 대해 y값을 계산!

plt.figure(figsize=(6.4, 4.8)) # 그림 그릴 도화지 크기를 정할 수도 있어! (가로 6.4인치, 세로 4.8인치)
plt.plot(x_values, y_values, c="green", linewidth=2, alpha=0.7) # 초록색, 두께 2, 투명도 0.7 선!
plt.scatter(x_values, y_values, s=70, c="red", alpha=0.5) # 점도 함께 빨갛게!
plt.grid(True) # 그림에 모눈종이처럼 격자도 그려주고!
plt.xlim(-10, 10) # x축은 -10부터 10까지만 보여줘!
# plt.ylim(0, 110) # y축 범위를 직접 정할 수도 있지! (지금은 주석 처리)
plt.show()
```

코딩새싹 🌱: `map(lambda ...)` 로 y값들을 한 번에 계산해서 `plt.plot()`에 넣어주니까 정말 함수 그래프가 그려졌어요! `plt.grid(True)`는 모눈종이 같아서 보기 편하네요!

<3. 그림을 더 아름답게! 그래프 꾸미기 마법!>

선생님: 우리가 그린 그림에 제목도 붙이고, x축과 y축이 무엇을 의미하는지 설명도 써주고, 범례도 달아주면 훨씬 더 멋진 작품이 되겠지?

```
# 3x + 2 (이차함수 예시를 1차 함수로 변경하신 듯하여 y = x^2 + 2로 진행)
# 그리고 y = 1/2 * x + 2 그래프를 함께 그려보자!
x = list(range(-10, 11)) # x좌표들 (리스트로 변환)
y1 = list(map(lambda i: i**2 + 2, x))  # y = x^2 + 2
y2 = list(map(lambda i: (1/2) * i + 2, x)) # y = 0.5x + 2

plt.figure(figsize=(6, 6)) # 도화지 크기는 6x6 정사각형으로!

# 첫 번째 그래프: 점과 선을 함께, 라벨도 붙여주자!
plt.scatter(x, y1, c='g', s=50, alpha=0.5, label='점 (x^2 + 2)')
plt.plot(x, y1, c='blue', alpha=0.5, label='선 (x^2 + 2)')
# 두 번째 그래프: 빨간 선으로, 라벨도 붙여주자!
plt.plot(x, y2, c='red', label='선 (0.5x + 2)')

plt.grid(True) # 격자 표시
plt.legend(fontsize=10, loc='upper center') # 범례 표시! 글자 크기 10, 위치는 위쪽 중앙!
                                            # loc 옵션 (0~10 또는 문자열)으로 위치 조절 가능!

# x축, y축 눈금 꾸미기
plt.xticks(fontsize=15, color='blue')
plt.yticks(fontsize=10, color='darkgreen') # y축 눈금은 작고 진한 초록색으로!

# 제목과 축 라벨 달기
plt.title("두 함수 그래프의 만남", fontsize=30, color='purple')
```

```
plt.xlabel("햇빛의 양 (Sun light)", fontsize=20, color='chocolate')
plt.ylabel("사과의 당도 (Apple sweet)", fontsize=20, color='magenta')

# 특정 위치에 가로/세로선 그리기
# plt.axvline(x=5, c='red', linewidth=3) # x=5 위치에 세로선 (주석 처리)
plt.axhline(y=10, color='gray', linestyle='--') # y=10 위치에 회색 점선 가로선!
plt.show()
```

궁금이 💡: `label='...'` 이렇게 각 그래프에 이름을 붙여주고 `plt.legend()`를 부르니까 어떤 선이 어떤 그래프인지 알려주는 범례가 생겼어요! `loc`으로 위치도 바꿀 수 있군요! (선생님이 범례 위치 코드 목록을 보여준다)

<4. 넘파이와 함께 그리는 여러 개의 직선들!>

선생님: 넘파이 배열을 사용하면 데이터를 만들고 계산하기가 훨씬 편하다고 했지? 여러 개의 직선 그래프를 `for` 반복문으로 한 번에 그려볼 수도 있어!

```
# 복습: 파이썬 리스트와 넘파이 배열의 덧셈 차이!
# a_list = [1, 2, 3]; # print(a_list + 2) # 이건 에러!
# b_np = np.array(a_list); print(b_np + 2) # 넘파이 배열은 각 항목에 2를 더해줘! [3 4 5]

x_np = np.arange(-10, 10) # -10부터 9까지의 정수 넘파이 배열!
plt.figure(figsize=(6, 5))
plt.title("여러 가지 기울기의 직선들", fontsize=20)

slopes = [-2, -1, 1, 2] # 기울기 후보들
for i in slopes:
  plt.plot(x_np, i * x_np, label=f"y = {i}*x") # y = ix 그래프 그리기!

plt.grid(True)
plt.legend(fontsize=12)
plt.axvline(0, color="black", linewidth=2) # x=0 (y축) 선을 더 굵게!
plt.axhline(0, color="black", linewidth=2) # y=0 (x축) 선을 더 굵게!
plt.show()
```

코딩새싹 🌱: `i * x_np` 하니까 `x_np` 배열의 모든 값에 `i`가 곱해져서 y값들이 한 번에 계산되네요! 넘파이 정말 편리해요! `for`문으로 여러 직선을 그리고 `label`을 다르게 주니까 범례도 예쁘게 나와요!

[도전!] y = x + b 형태의 직선들을 그려보고, y절편 (0, b)에 점과 텍스트도 표시해보세요!

```
x_np = np.arange(-10, 10)
y_intercepts = np.arange(-5, 5, 2) # -5, -3, -1, 1, 3 (y절편 후보들)

plt.figure(figsize=(7,7))
```

```
plt.title("다양한 y절편을 가진 직선들", fontsize=20)

for b_val in y_intercepts:
  plt.plot(x_np, x_np + b_val, label=f"y = x + {b_val}")
  plt.scatter(0, b_val, s=100, c='red') # y절편 (0, b_val)에 빨간 점 찍기!
  plt.text(0 + 0.3, b_val, f"(0, {b_val})", fontsize=9) # 점 옆에 좌표 텍스트 쓰기!

plt.grid(True)
plt.legend()
plt.axvline(0, color="black", linewidth=2)
plt.axhline(0, color="black", linewidth=2)
plt.show()
```

<5. 특별한 곡선, 분수 함수 그리기 (y = 1/x)!>

선생님: 이번엔 조금 특별한 함수, y = 1/x 그래프를 그려볼까? 이 함수는 x가 0일 때 문제가 생기는데, 넘파이와 Matplotlib은 어떻게 처리할까?

```
x_hyperbola = np.arange(-10, 11) # -10부터 10까지. 중간에 0이 포함돼!
# x_hyperbola = np.linspace(-10, 10, 100) # 더 부드러운 곡선을 위해 촘촘한 x값 사용 가능!
# x_hyperbola = x_hyperbola[x_hyperbola != 0] # 0을 제외하고 싶다면 이렇게!

plt.figure(figsize=(5, 6))
plt.title('y = 1/x 그래프', fontsize=20)

# 넘파이는 x가 0일 때 1/0 계산에서 에러 대신 '경고'를 내고 무한대(inf) 값을 사용해.
# Matplotlib은 이 무한대 값을 보통 화면 밖으로 그려서 그래프가 끊어진 것처럼 보여줘.
y_hyperbola = 1 / x_hyperbola

plt.scatter(x_hyperbola, y_hyperbola)
plt.plot(x_hyperbola, y_hyperbola)
plt.axvline(0, color="black", linewidth=2)
plt.axhline(0, color="black", linewidth=2)
plt.grid(True)
plt.ylim(-10, 10) # y축 범위를 제한해서 너무 큰 값은 안 보이게!
plt.show()
```

궁금이 💡: x가 0일 때는 0으로 나눌 수 없는데, 에러가 안 나고 그래프가 그려지긴 하네요! 대신 0 근처에서 선이 쭉 뻗어나가다가 끊어지는 것 같아요!

선생님: 맞아! 넘파이는 0으로 나누는 상황에서 프로그램이 멈추는 대신 '무한대(inf)'라는 특별한 값을 사용하고, Matplotlib은 이 부분을 보통 화면 바깥으로 그려내서 그래프가 x=0 점근선에서 끊어지는 것처럼 표현한단다.

[도전!] y = 1/x, y = 2/x, y = 3/x 그래프를 한 번에 그려보고 범례도 표시해보세요!

```
numerators = [1, 2, 3]
x_dense = np.linspace(-10, 10, 500) # 더 부드러운 곡선을 위해 촘촘한 x값!
x_dense = x_dense[x_dense != 0] # 0을 제외!

plt.figure(figsize=(6, 8))
plt.title('여러 가지 분수 함수 그래프', fontsize=20)

for k in numerators:
  y_vals = k / x_dense
  plt.plot(x_dense, y_vals, label=f"y = {k}/x")

plt.legend()
plt.grid(True)
plt.axvline(0, color="black", linewidth=1)
plt.axhline(0, color="black", linewidth=1)
plt.ylim(-10, 10) # y축 범위 제한
plt.show()
```

선생님: (만족스러운 표정으로) 자, 오늘 우리는 Matplotlib이라는 멋진 화가를 만나서 점과 선으로 다양한 이야기를 그려봤어! 간단한 점 찍기부터 복잡한 함수 그래프, 그리고 여러 그래프를 한 번에 그리고 예쁘게 꾸미는 방법까지! 이제 여러분은 숫자 데이터를 눈으로 보는 즐거움과 강력한 분석 도구를 함께 얻게 된 거야! 앞으로 이 시각화 마법으로 어떤 멋진 이야기들을 그려낼지 정말 기대되는구나! 🎨🌟