
نازیلا سپهرکیا (Nazila Sepehr Kiya)،

- ✓ صاحب امتیاز و مدیر مسوول سایت اطلاع رسانی آموزشی – پژوهشی زیستگاه مدرن
- ✓ نویسنده کد خطر سیلاب شهری برای شهر زوریخ و تشریح عملکرد کدها در چند قسمت به صورت سریالی
- ✓ نحوه عملکرد کدهای ماژول hazard مربوط به سیلاب شهری، مورد مطالعه شهر زوریخ در سوئیس، که در گیت هاب من به آدرس:https://github.com/sepehrkiya/climada_petal_urban_flood
- ✓ ارائه شده است، در چند قسمت توضیح داده خواهد شد.
- ✓ توجه: این کدها به صورت منبع باز نوشته شده و اجازه کپی برداری برای عموم آزاد است. تلاش بر این است که کدها در محیط Climada قابل اجرا باشند.

قسمت اول :

تعریف کلاسِ ZurichFloodHazard:

این کلاس با استفاده از یک مدل خطر سیل، پیش‌بینی حوادث سیلاب را در شهر زوریخ امکان‌پذیر می‌کند.

```
class ZurichFloodHazard(Hazard):
  def __init__(self):
    """Initialize Zurich flood hazard model with default parameters."""
    super().__init__(haz_type='FL')
    self.units = 'm'
    self.pool = None
    self.rainfall_data = None
    self.dem_data = None
    self.districts = None
    self.drainage_network = None
    self.soil_properties = None
    self.lake_zurich = None
    self.rivers = None
    self.model_params = {
      'dx': 10.0,  # Grid resolution in meters
      'dt': 1.0,   # Time step in seconds
      'g': 9.81,   # Gravity acceleration
      'manning_n': 0.03,  # Manning roughness coefficient
      'theta': 0.7,  # Time integration parameter (0.5-1.0)
      'max_iter': 10000,  # Maximum iterations
      'convergence_threshold': 1e-5,  # Convergence criterion
      'cfl_number': 0.7,  # Courant–Friedrichs–Lewy condition
      'output_interval': 3600,  # Output interval in seconds
      'sim_duration': 86400,  # Simulation duration in seconds
      'infiltration': {
        'initial_moisture': 0.3,  # Initial moisture content
        'saturated_moisture': 0.5,  # Saturated moisture content
        'suction_head': 0.1667,  # Suction head in meters
        'hydraulic_conductivity': {
          'urban': 5e-7,     # Urban areas (m/s)
```

```
            'suburban': 1e-6,   # Suburban areas (m/s)
            'parks': 5e-6,      # Parks and gardens (m/s)
            'forest': 1e-5      # Forest areas (m/s)
        }
    },
    'drainage': {
        'capacity': 0.02,  # Base drainage capacity in m/h
        'upgrade_factor': {  # Capacity multiplier by district
            'district_1': 1.2,  # Central district
            'district_2': 1.0,
            'district_3': 0.9,
            'district_4': 1.1,
            'district_5': 1.0,
            'district_6': 0.8,
            'district_7': 1.1,
            'district_8': 1.0,
            'district_9': 0.9,
            'district_10': 0.7, # Newest development
            'district_11': 1.0,
            'district_12': 0.8
        }
    }
}
```

با این کلاس، که به نام `ZurichFloodHazard` نامگذاری کرده ایم، یک مدل خطر سیل آبزی برای شهر زوریخ ایجاد می کنیم.

این کلاس با استفاده از توابع و پارامترهای پیش‌فرض شده در آن، قابلیت پیش‌بینی حوادث سیلاب را دارد.

در این کلاس، برخی از پارامترها بر اساس اطلاعات زمینی، داده‌های بارش، شبکه آبراهه ها و ویژگی‌های زمین تنظیم می‌شوند. همچنین، مدل نیز توانایی تولید گرافیک نتیجه را برای هر واحد زمانی دارا می باشد.

از نظر فنی، این کلاس با استفاده از یک تابع `__init__` خود را در کلاس پدر (Hazard) پیاده‌سازی کرده است. همچنین، پارامترهای پیش‌فرض توسط این کلاس شامل واحدهای اندازه‌گیری، داده‌های زمینی و سایر اطلاعات ضروری برای مدل خطر سیل هستند.

از نظر عملکرد، این کلاس از یک تابع `__init__` که در آن پارامترهای پیش‌فرض شده به همراه تنظیمات لازم برای مدل خطر سیل نوشته شده، پیاده‌سازی شده است. همچنین، سایر توابع مانند تنظیمات لازم برای حوادث سیلاب را نیز ارائه کرده است.

حال به تحلیل خطوط این کد می پردازیم:

```
class ZurichFloodHazard(Hazard):
```

```
    def __init__(self):
```

این خط کد کلاس `ZurichFloodHazard` را به کلاس پدر (Hazard) متصل ساخته و کلاس خود را با پارامترهای پیش‌فرض شده در آن توصیف میکند.

```
super().__init__(haz_type='FL')
        self.units = 'm'
```

```
self.pool = None
self.rainfall_data = None
self.dem_data = None
self.districts = None
self.drainage_network = None
self.soil_properties = None
self.lake_zurich = None
self.rivers = None
```

در خطوط فوق ، واحد اندازه‌گیری برای مدل خطر سیل به عنوان متر (م) تنظیم می‌شود و همچنین، نوع خطر (haz_type) نیز بر اساس مدل خطر سیل آبزی (FL) در نظر گرفته شده است.

سپس پارامترهای پیش‌فرض شده برای مدل خطر سیل با نام‌های مختلف مانند داده‌های بارش (rainfall_data)، شبکه‌های آبراهه (drainage_network)، داده‌های زمینی (dem_data) و غیره به عنوان اشیا مطرح می‌شوند.

```
self.model_params = {
    'dx': 10.0,  # Grid resolution in meters
    'dt': 1.0,   # Time step in seconds
    'g': 9.81,   # Gravity acceleration
    'manning_n': 0.03,  # Manning roughness coefficient
    'theta': 0.7,  # Time integration parameter (0.5-1.0)
    'max_iter': 10000,  # Maximum iterations
    'convergence_threshold': 1e-5,  # Convergence criterion
    'cfl_number': 0.7,  # Courant–Friedrichs–Lewy condition
    'output_interval': 3600,  # Output interval in seconds
    'sim_duration': 86400,  # Simulation duration in seconds
    'infiltration': {
        'initial_moisture': 0.3,  # Initial moisture content
        'saturated_moisture': 0.5,  # Saturated moisture content
        'suction_head': 0.1667,  # Suction head in meters
        'hydraulic_conductivity': {
            'urban': 5e-7,     # Urban areas (m/s)
            'suburban': 1e-6,   # Suburban areas (m/s)
            'parks': 5e-6,     # Parks and gardens (m/s)
            'forest': 1e-5     # Forest areas (m/s)
        }
```

و در نهایت پارامترهای پیش‌فرض شده برای مدل خطر سیل در یک دیکشنری (model_params) ذخیره می‌شوند که شامل پارامترهای گوناگونی مانند نرخ تصحیح (infiltration)، ظرفیت آبراهه ها (drainage) بر اساس منطقه شهری مورد نظر و غیره هستند.

```
    'drainage': {
        'capacity': 0.02,  # Base drainage capacity in m/h
        'upgrade_factor': {  # Capacity multiplier by district
            'district_1': 1.2,  # Central district
            'district_2': 1.0,
```

```
            'district_3': 0.9,
            'district_4': 1.1,
            'district_5': 1.0,
            'district_6': 0.8,
            'district_7': 1.1,
            'district_8': 1.0,
            'district_9': 0.9,
            'district_10': 0.7, # Newest development
            'district_11': 1.0,
            'district_12': 0.8
        }
```

حال نوبت به مرحله بارگذاری داده های بارندگی از مخازن اقلیمی می رسد و اینکار را با تعریف یک متد برای کلاس `ZurichFloodHazard` انجام می دهیم.

نام این متد را `set_from_copernicus_cds` انتخاب میکنیم و ظیفه بارگذاری داده های بارندگی از مخزن اقلیمی کوپرنیک[1] را به آن می سپاریم:

```
def set_from_copernicus_cds(self, start_date, end_date, api_key=None):
    """Load precipitation data from Copernicus Climate Data Store.

    Parameters:
        start_date (str): Start date in format 'YYYY-MM-DD'
        end_date (str): End date in format 'YYYY-MM-DD'
        api_key (str, optional): Copernicus CDS API key

    Returns:
        ZurichFloodHazard: self
    """
    # Check if API key is provided or exists in environment
    if api_key is None:
        api_key = os.environ.get('COPERNICUS_CDS_API_KEY')
        if api_key is None:
            raise ValueError("No Copernicus CDS API key provided")

    # Format request to Copernicus CDS
    url = "https://cds.climate.copernicus.eu/api/v2"
    request_params = {
        'dataset_short_name': 'reanalysis-era5-land',
        'product_type': 'reanalysis',
        'format': 'netcdf',
        'variable': ['total_precipitation'],
        'year': [d.split('-')[0] for d in [start_date, end_date]],
        'month': list(range(1, 13)),
```

[1] Copernicus (Copernicus Climate Data Store)

```
    'day': list(range(1, 32)),
    'time': [f"{h:02d}:00" for h in range(24)],
    'area': [47.4, 8.4, 47.3, 8.6],  # Zurich bounding box
}

# Log request details
print(f"Requesting precipitation data from Copernicus CDS for {start_date} to {end_date}")

# This is a simplified placeholder - actual implementation would use the CDS API client
# For demonstration only - in production code, use proper CDS API client
# self.rainfall_data = xr.open_dataset(response_file)

# Placeholder simulated data (in actual implementation, this would come from CDS)
time_range = pd.date_range(start=start_date, end=end_date, freq='1H')
lats = np.linspace(47.3, 47.4, 20)
lons = np.linspace(8.4, 8.6, 20)
# Create synthetic precipitation data with realistic spatio-temporal pattern
# Rainstorm events with higher intensity in summer months
precip_data = np.zeros((len(time_range), len(lats), len(lons)))

for i, t in enumerate(time_range):
    month = t.month
    hour = t.hour

    # More intense rain in summer (May-September)
    seasonal_factor = 2.0 if 5 <= month <= 9 else 0.8

    # Diurnal cycle - more rain in afternoon/evening
    diurnal_factor = 1.5 if 14 <= hour <= 20 else 0.9

    # Random storm events (approximately 10% of time steps)
    if np.random.random() < 0.1:
        storm_center_lat = np.random.choice(range(len(lats)))
        storm_center_lon = np.random.choice(range(len(lons)))

        # Create storm pattern
        for lat_idx in range(len(lats)):
            for lon_idx in range(len(lons)):
                distance = np.sqrt((lat_idx - storm_center_lat)**2 +
                                   (lon_idx - storm_center_lon)**2)
                intensity = 10 * np.exp(-0.5 * (distance/3)**2) * seasonal_factor * diurnal_factor
                precip_data[i, lat_idx, lon_idx] = max(0, intensity + np.random.normal(0, 0.2))
    else:
        # Background precipitation
        precip_data[i, :, :] = np.random.exponential(0.5) * seasonal_factor * diurnal_factor
```

```
# Create xarray dataset
self.rainfall_data = xr.Dataset(
    data_vars={
        "tp": (["time", "latitude", "longitude"], precip_data)
    },
    coords={
        "time": time_range,
        "latitude": lats,
        "longitude": lons
    }
)
```

وظایف این کد را در چند بخش می نویسیم:

۱. چک کردن API Key: در این مرحله، برنامه بررسی می‌کنیم که آیا API Key برای مخزن اقلیمی Copernicus وجود دارد یا خیر؟

اگر موجود نیست، برنامه خطای ورودی را ایجاد می‌کند.

۲. شکل دهی درخواست: در این مرحله، برنامه شکل دهی؛ درخواست خود را به مخزن اقلیمی Copernicus می فرستد و پارامترهای لازم برای بارگیری داده‌های بارندگی در نظر گرفته می شود.

درخواست شامل پارامترهایی مثل نام دیتا سِت، نوع محصول، فرمت، متغیرها، سال، ماه، روز و زمان است.

```
    # Check if API key is provided or exists in environment
    if api_key is None:
        api_key = os.environ.get('COPERNICUS_CDS_API_KEY')
        if api_key is None:
            raise ValueError("No Copernicus CDS API key provided")
```

۳. درخواست‌دادن: در این مرحله، کد های برنامه برای دریافت داده‌های بارندگی از مخزن اقلیمی Copernicus را می نویسیم:

```
# Format request to Copernicus CDS
    url = "https://cds.climate.copernicus.eu/api/v2"
    request_params = {
        'dataset_short_name': 'reanalysis-era5-land',
        'product_type': 'reanalysis',
        'format': 'netcdf',
        'variable': ['total_precipitation'],
        'year': [d.split('-')[0] for d in [start_date, end_date]],
        'month': list(range(1, 13)),
        'day': list(range(1, 32)),
        'time': [f"{h:02d}:00" for h in range(24)],
        'area': [47.4, 8.4, 47.3, 8.6],  # Zurich bounding box
    }

    # Log request details
```

```
print(f"Requesting precipitation data from Copernicus CDS for {start_date} to {end_date}")
```

٤. طراحی ساختار برای بارگذاری داده: در این مرحله، ساختاری را برای بارگذاری داده‌های بارندگی ایجاد می کنیم و با داده‌های مصنوعی که در مخزن اقلیمی Copernicus وجود دارد مقایسه می‌کنیم.

```
# Placeholder simulated data (in actual implementation, this would come from CDS)
    time_range = pd.date_range(start=start_date, end=end_date, freq='1H')
    lats = np.linspace(47.3, 47.4, 20)
    lons = np.linspace(8.4, 8.6, 20)
```

٥. بارگذاری داده های بارندگی: در این مرحله، بارگذاری داده‌های بارندگی به فرمت xarray انجام می‌شود.

```
precip_data = np.zeros((len(time_range), len(lats), len(lons)))
```

این کد بارگذاری داده‌های بارندگی را به فرمت xarray انجام می‌دهد. سه‌بند برای داده‌های بارندگی در نظر گرفته شده است که شامل زمان، عرض جغرافیایی و طول جغرافیایی هستند.

و بطور خلاصه، متد فوق یعنی: `set_from_copernicus_cds`، وظیفه بارگذاری داده‌های بارندگی از مخزن اقلیمی Copernicus را بر عهده دارد. این شامل چک کردن API_key، فرمت‌سازی درخواست، ساختار برای بارگذاری داده و بارگذاری داده است.

بعد به بررسی حلقه for می پردازیم :

در این حلقه بازه زمانی بارندگی را بررسی می کنیم . زمان را بر اساس ماه و ساعت تعریف می کنیم و برای هریک از این زمانها شرط می گذاریم.

```
for i, t in enumerate(time_range):
    month = t.month
    hour = t.hour

    # More intense rain in summer (May-September)
    seasonal_factor = 2.0 if 5 <= month <= 9 else 0.8

    # Diurnal cycle - more rain in afternoon/evening
    diurnal_factor = 1.5 if 14 <= hour <= 20 else 0.9
```

اعمال شرط برای رویدار طوفان را در این خط انجام می دهیم:

```
    # Random storm events (approximately 10% of time steps)
    if np.random.random() < 0.1:
        storm_center_lat = np.random.choice(range(len(lats)))
        storm_center_lon = np.random.choice(range(len(lons)))
```

و سپس در اینجا الگوی طوفان مصنوعی را با استفاده از پارامترهای زمانی ایجاد می کنیم و برای حالتهای مختلف آن محاسبات لازم را انجام میدهیم :

```
        # Create storm pattern
        for lat_idx in range(len(lats)):
            for lon_idx in range(len(lons)):
```

سپس فاصله بین دو نقطه مرکز حادثه و مرکز طوفان را محاسبه می کنیم

```
distance = np.sqrt((lat_idx - storm_center_lat)**2 +
                   (lon_idx - storm_center_lon)**2)
```

حال نوبت به محاسبه شدت بارندگی می رسد

```
intensity = 10 * np.exp(-0.5 * (distance/3)**2) * seasonal_factor * diurnal_factor
precip_data[i, lat_idx, lon_idx] = max(0, intensity + np.random.normal(0, 0.2))
```

این کد مقدار باران باریده شده را بر اساس شدت باران و یک عامل تصادفی، تایید می کند و اگر شدت باران مقدار مثبتی باشد مقدار باران هم مثبت در نظر گرفته می شود. همچنین چک میکند که آیا حادثه اتفاق افتاده است یا خیر ؟
در غیر اینصورت :

```
    else:
        # Background precipitation
        precip_data[I, :, :] = np.random.exponential(0.5) * seasonal_factor * diurnal_factor
```

بر اساس نقاط جغرافیایی، باران مصنوعی را با استفاده از پارامترهای زمانی و روزانه درست می کنیم.

```
    # Create xarray dataset
    self.rainfall_data = xr.Dataset(
        data_vars={
            "tp": (["time", "latitude", "longitude"], precip_data)
        },
        coords={
            "time": time_range,
            "latitude": lats,
            "longitude": lons
        }
    )
```

حالا نوبت به تولید داده های توصیفی می رسد:

```
self.rainfall_data.tp.attrs["units"] = "mm/h"
    self.rainfall_data.tp.attrs["long_name"] = "Total Precipitation"
        print(f"Successfully loaded precipitation data with {len(time_range)} timestamps")
return self
```

و سپس به بارگذاری داده های شهر زوریخ می پردازیم:

```
def load_zurich_data(self, data_dir=None):
    if data_dir is None:
        data_dir = os.path.join(SYSTEM_DIR, 'data', 'zurich')

    # In a real implementation, this would load actual data files
    # For demonstration, we'll create synthetic data

    # Create synthetic DEM with realistic Zurich topography
    # Zurich has elevation ranging from ~400m to ~850m
```

```
grid_size = 500
x = np.linspace(0, 10000, grid_size)  # 10 km grid
y = np.linspace(0, 10000, grid_size)  # 10 km grid

# Base elevation
base_elevation = 400 + np.zeros((grid_size, grid_size))

# Lake Zurich (south-east)
lake_mask = np.zeros((grid_size, grid_size), dtype=bool)
```

۱. اگر پارامتر `data_dir` ورودی باشد، از آن استفاده می‌کند. اگر خالی باشد (یعنی ورودی نشده باشد)، مسیر پیش فرض برای ذخیره‌سازی داده‌های زوریخ را تعیین می‌کند.
۲. با توجه به اینکه یک محیط شبیه‌سازی است، داده‌های واقعی، جایگزین داده‌های تولیدی خواهند شد.

خط `if data_dir is None:`
اگر `data_dir` خالی باشد (بدین معناست که هیچ ورودی داده نشده)، مسیر پیش‌فرض برای ذخیره‌سازی داده‌ها را تعیین می‌کند. این خط ممکن است در شرایط خاصی، مثلاً در محیط توسعه با توجه به نیاز به استفاده از داده‌های زوریخ، به کار برده نشود.

```
if data_dir is None:
    data_dir = os.path.join(SYSTEM_DIR, 'data', 'zurich')
```

۳. کد `grid_size = 500` مسوولیت تعیین اندازه‌گیری برای ایجاد یک grid شبیه‌سازی شده برای DEM را بر عهده دارد.

```
grid_size = 500
x = np.linspace(0, 10000, grid_size)  # 10 km grid
y = np.linspace(0, 10000, grid_size)  # 10 km grid
```

٤. کد base_elevation

```
base_elevation = 400 + np.zeros((grid_size, grid_size))
```

در این خط، یک ارتفاع پایه با توجه به ارتفاع دهکده‌ها در زوریخ ایجاد شده است. اگر این ارتفاع پایه بیشتر از حد باشد، ممکن است در این خط، خطا ایجاد شود.

۵. در خط بعدی یعنی :

```
lake_mask = np.zeros((grid_size, grid_size), dtype=bool)
```

یک مکان‌سنجی برای دریاچه‌های زوریخ به کار گرفته شده است. اگر این مکان‌سنجی اشتباه باشد یا خالی باشد، در این خط نیز ممکن است خطا ایجاد شود.

بنابراین نیاز به قرار دادن یک حلقه با شرطهای مورد نیاز خواهد بود:

```
for i in range(grid_size):
    for j in range(grid_size):
        if (i-grid_size*0.7)**2 + (j-grid_size*0.7)**2 < (grid_size*0.3)**2:
            lake_mask[i, j] = True
```

```
        base_elevation[lake_mask] = 399  # Lake level
        # Hills (Uetliberg in west, Zürichberg in east)
    xx, yy = np.meshgrid(x, y)
        # Uetliberg (west side)
    uetliberg = 400 * np.exp(-0.5 * ((xx-2000)**2 + (yy-5000)**2) / 2000**2)
        # Zürichberg (east side)
    zurichberg = 300 * np.exp(-0.5 * ((xx-8000)**2 + (yy-5000)**2) / 2000**2)
        # Combining topographic features
    self.dem_data = base_elevation + uetliberg + zurichberg
        # Synthetic river path (Limmat river running from lake through city center)
    river_y = np.linspace(grid_size*0.7, 0, 100)
    river_x = grid_size*0.7 + np.sin(np.linspace(0, np.pi*2, 100)) * grid_size*0.05
        # Carve river into DEM
    for i, (x, y) in enumerate(zip(river_x, river_y)):
      x_idx = int(x)
      y_idx = int(y)
      if 0 <= x_idx < grid_size and 0 <= y_idx < grid_size:
        # River channel with declining elevation from lake to north
        river_elevation = 398 - i*0.1
        # Create a river bed approx 30m wide
        for dx in range(-15, 16):
          for dy in range(-15, 16):
            if x_idx+dx >= 0 and x_idx+dx < grid_size and y_idx+dy >= 0 and y_idx+dy < grid_size:
              dist = np.sqrt(dx**2 + dy**2)
              if dist < 15:
                self.dem_data[y_idx+dy, x_idx+dx] = min(
                  self.dem_data[y_idx+dy, x_idx+dx],
                  river_elevation + dist*0.2
                )
```

این کد برای ایجاد یک مدل دیجیتال از زمین (DEM) برای شهر زوریخ استفاده می‌شود. این مدل شامل آب و هوای طبیعی، تپه‌ها، دریاچه‌ها و حتی مسیرهای رودخانه نیز می‌شود.

```
for i in range(grid_size):
```

```
    for j in range(grid_size):
        if (i-grid_size*0.7)**2 + (j-grid_size*0.7)**2 < (grid_size*0.3)**2:
            lake_mask[i, j] = True
        base_elevation[lake_mask] = 399  # Lake level
```

شرح وظایف کدهای داخل حلقه ذکر شده به شرح زیر است:

```
lake_mask = np.zeros((grid_size, grid_size), dtype=bool)
```

این خط کد برای ایجاد یک پوشه (ماسک) استفاده می‌شود. در ابتدا به یک مساحتی با اندازه `grid_size` مقداردهی می‌شوند که تمام مقادیر آنها صفر است.
این حلقه روی هر نقطه ازgrid از مساحتی با اندازه `grid_size` اجرا می‌شود.
یکبار برای سایز گرید i اُم و بار دیگر برای سایز گرید jاُم عملیات انجام می شود.
اگر شرط اول برآورده شود، مقادیر این مساحتی که در داخل حلقه هستند، به True تغییر می یابند.
در داخل این حلقه، برای هر نقطه، یک شرط وجود دارد:
اگر

```
if (i-grid_size*0.7)**2 + (j-grid_size*0.7)**2 < (grid_size*0.3)**2:
    lake_mask[i, j] = True
```

با استفاده از ماسک ساخته شده قبلی، تمام نقاطی که در آنها آب دریاچه قرار دارد، ارتفاعشان برابر ۳۹۹ تنظیم می‌شود.

```
base_elevation[lake_mask] = 399  # Lake level
```

نوبت به شرطهای داخل این حلقه می رسد:

```
xx, yy = np.meshgrid(x, y)
        # Uetliberg (west side)
    uetliberg = 400 * np.exp(-0.5 * ((xx-2000)**2 + (yy-5000)**2) / 2000**2)
        # Zürichberg (east side)
    zurichberg = 300 * np.exp(-0.5 * ((xx-8000)**2 + (yy-5000)**2) / 2000**2)
     # Combining topographic features
    self.dem_data = base_elevation + uetliberg + zurichberg
    # Synthetic river path (Limmat river running from lake through city center)
    river_y = np.linspace(grid_size*0.7, 0, 100)
    river_x = grid_size*0.7 + np.sin(np.linspace(0, np.pi*2, 100)) * grid_size*0.05
```

`xx, yy = np.meshgrid(x, y)` و شرط‌های داخل این حلقه : به اشکال مقدار اختصاص یافته و به داده تبدیل می‌شوند.
`uetliberg` و شرط‌های داخل این حلقه: ارتفاع تپه‌ای که در قسمت غرب شهر قرار دارد، را برآورد می‌کنند.
`zurichberg` و شرط‌های داخل این حلقه : ارتفاع تپه‌ای که در قسمت شرق شهر قرار دارد، را برآورد می‌کنند.

در خط: `self.dem_data = base_elevation + uetliberg + zurichberg`، تمام ارتفاعات شهر از آب دریاچه، تپه‌ها و دریاچه‌ها برآوردن شده‌اند.
شرط‌های داخل حلقه `river_x`: ارتفاع مسیر رودخانه‌ای که در آن جاری است، را برآورد می‌کنند.
و در کدهای زیر، رودخانه به دِم تبدیل شده است:

```
for i, (x, y) in enumerate(zip(river_x, river_y)):
        x_idx = int(x)
        y_idx = int(y)
        if 0 <= x_idx < grid_size and 0 <= y_idx < grid_size:
            # River channel with declining elevation from lake to north
            river_elevation = 398 - i*0.1
            # Create a river bed approx 30m wide
```

```
for dx in range(-15, 16):
    for dy in range(-15, 16):
        if x_idx+dx >= 0 and x_idx+dx < grid_size and y_idx+dy >= 0 and y_idx+dy < grid_size:
            dist = np.sqrt(dx**2 + dy**2)
            if dist < 15:
                self.dem_data[y_idx+dy, x_idx+dx] = min(
                    self.dem_data[y_idx+dy, x_idx+dx],
                    river_elevation + dist*0.2
                )
```

این شرط دقیقا خط به خط موارد زیر را اجرا می کند:

۱. مرحله اول: در این شرط، یک زوج مختصات (`x`, `y`) از یک فهرست (`river_x` و `river_y`) را وارد می‌کند.
این زوج مختصات است و نشان‌دهنده موقعیت رودی در-grid است.

۲. مرحله دوم: اگر مختصات وارد شده صحیح باشند (مجموعه‌ای از عبارات شرطی) ایجاد می گردد.
مقادیر ارتفاع (`river_elevation`) بر اساس عدد آیتم (`i`) محاسبه می‌شود؛ هر آیتم، ارتفاع رودی را به در حال جاری شدن بدست می آورد
در واقع، این ارتفاع برابر با `۳۹۸ - i*0.1` است و در هر آیتم، یک نزول ۱/۰ اتفاق می‌افتد.
سپس یک فرسکو (`river_elevation`) ایجاد می‌شود که حدوداً ۳۰ متر عرض دارد (بیشترین اندازه)؛ که جریان رود را در محور `x` و `y` بر روی grid نمایش می‌دهد.

۳. مرحله سوم: در هر نقطه‌ای از grid که یک مقادیر ارتفاع (`river_elevation`) وجود دارد، یک فرسکو با عرض حدود ۳۰ متر ایجاد شده است. این فرسکو تقریباً در همه نقاطی که آن‌ها بر روی river واقع شده‌اند، با توجه به جاری شدن رود که در آن آیتم درنزدیکی از سرچشمه به شمال ایجاد خواهد شد.

٤. مرحله چهارم: یک نقطه‌ای (`self.dem_data[y_idx+dy, x_idx+dx]`) بر روی grid با توجه به فرسکو (`river_elevation + dist*0.2`) که در آن آیتم جریان رود اتفاق افتاده، مقدار ارتفاعش را بر اساس مقادیر عمقی که در آن نقطه واقع شده‌اند، تغییر می دهد.

در واقع این شرط نوشته شده ؛ یک فرسکو با عرض تقریباً ۳۰ متر در همه نقاطی که بر روی river (رود) واقع شده‌اند، ایجاد می کند و در هر نقطه‌ای از grid که یک مقادیر ارتفاع (`river_elevation`) وجود دارد، مقدار ارتفاعش را بر اساس مقادیر عمقی که در آن نقطه واقع شده‌اند، تغییر می‌دهد.

پایان قسمت اول (تاریخ تفسیر کد، ۱٤۰٤/۰۲/۱۲)