Python を使って作業の効率化を図ろう！

ESRIジャパン株式会社

目次

- Python の概要
  - Python とは
  - 開発環境/実行環境
- ArcPy とは
  - ArcPy とは
  - ArcPy サイト パッケージ
    - 関数
    - クラス
    - マッピング モジュール
    - データ アクセス モジュール
    - Spatial Analyst モジュール
  - ArcPy の使い方
    - ArcPy サイト パッケージ のインポート
- ArcPy を使った処理
  - マップ・レイヤーの操作
    - 基本フロー
    - オブジェクトの参照
      - プロジェクト ファイルの参照
      - マップの参照
      - レイヤーの参照
    - オブジェクトの操作
      - 表示の切り替え
    - マップ の出力
  - ジオプロセシング ツールの操作
    - スクリプトからのジオプロセシング ツール実行方法
  - データの操作
    - データの選択
    - Arcpy.da.cursor
    - リスト関数と Describe 関数

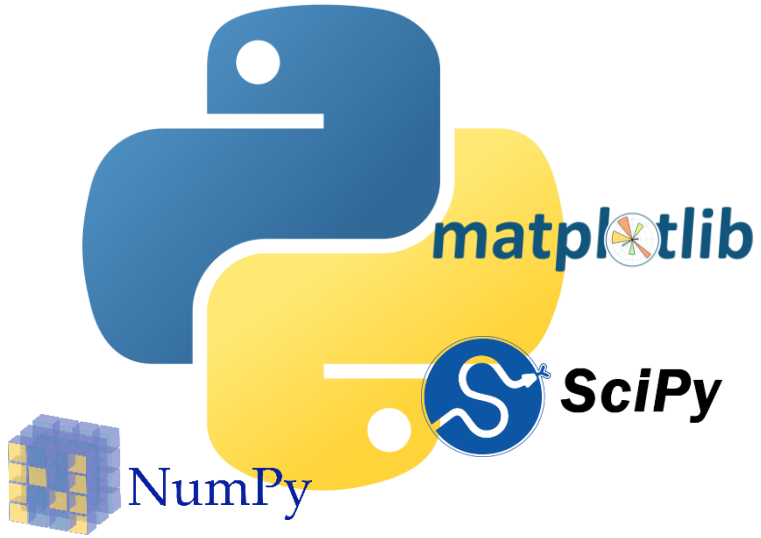
Python の概要

Python とは

- プログラミング言語
  - タスクを自動化する
  - シンプルなコーディングで可読性の高いコードが記述できる
  - 世界中で親しまれているスクリプト言語

- 豊富なライブラリが提供されている
  - 数学・科学技術計算ライブラリが使用できる
  - ビッグデータ解析や機械学習にも使用されている

- オープンソース
  - ArcGIS Desktop では、9.x のときから採用



Python とは
バージョンにおける留意点と ArcGIS

- 2 系と 3 系がある
  - ArcGIS Desktop (ArcMap)　：2系
  - ArcGIS Pro　：3系

- 書き方に違いがある場合がある
  - 3.x 系で Unicode に柔軟に対応できるようになった（日本語の扱いが容易になった）
  - すべてにおいて互換性があるわけではないので書き方には注意が必要！
    - 例） print (2系ではステートメントだったが、 3系では関数化された)

```
# Python 2系
print “出力する文字列”
```

```
# Python 3系
print(“出力する文字列”)
```

開発環境/実行環境

開発環境/実行環境

統合開発環境 (IDE)

- Python そのものは ArcGIS Desktop インストール時に同時にインストールされる
- IDE は、Python コードの記述とテストを行うツールを組み合わせたソフトウェア アプリケーション
- IDE 内でコードを記述し、コードの構文とコード補完を支援する機能を利用可能
- アプリケーション内からコードを実行することができ、IDE は結果ウィンドウにフィードバックを提供することが可能

開発環境/実行環境

参考情報

- 米国を含め Esri で利用されているもの
  - 2.x 系 : PyScripter

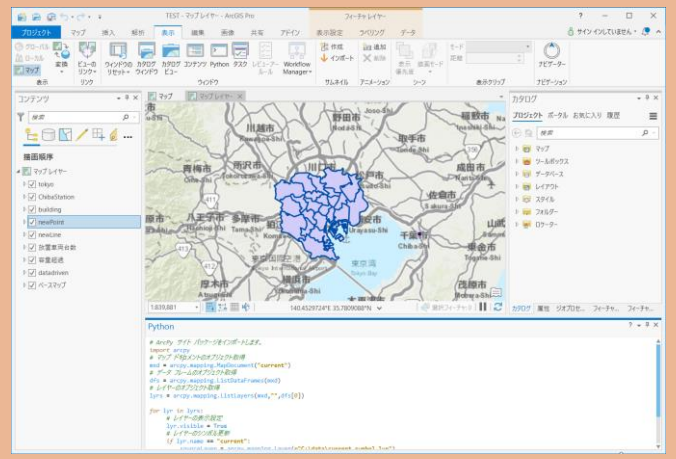
  - 3.x 系 : PyCharm,　Visual Studio,　VS Code

開発環境/実行環境

ArcPy を使った実行環境

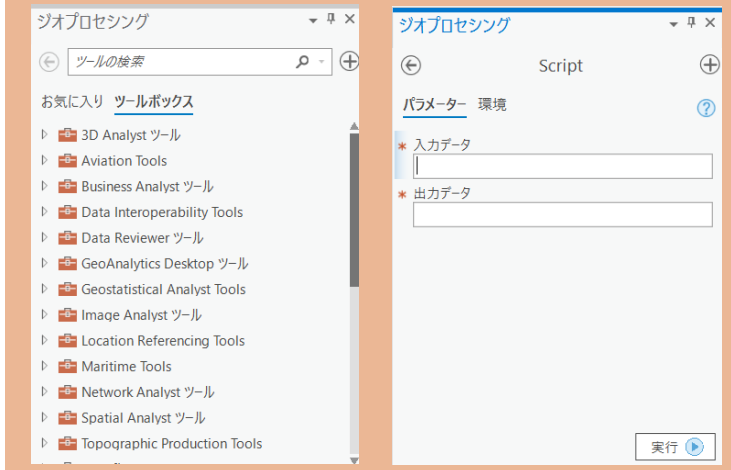
- 目的に応じて実行環境を選択

Python ウィンドウ



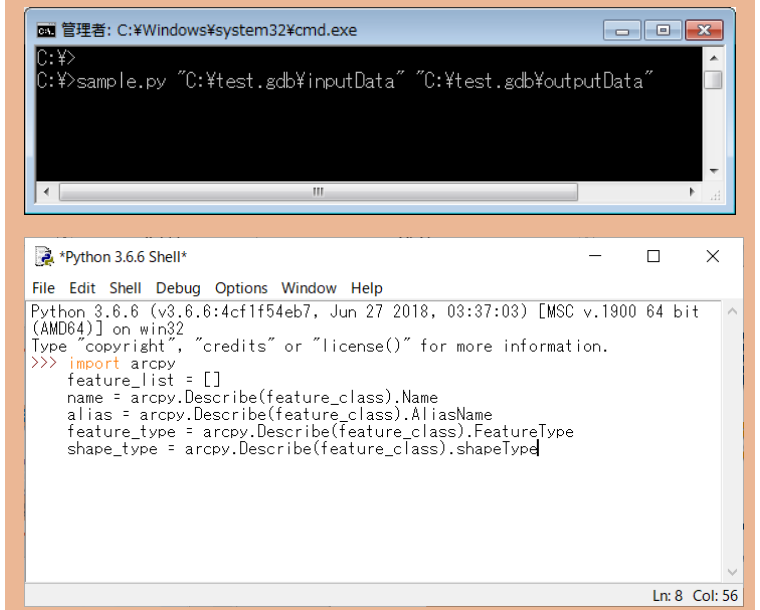
マップを見ながら
対話的に処理ができる

スクリプト ツール



ジオプロセシング
ツールとして使える

コマンドツール



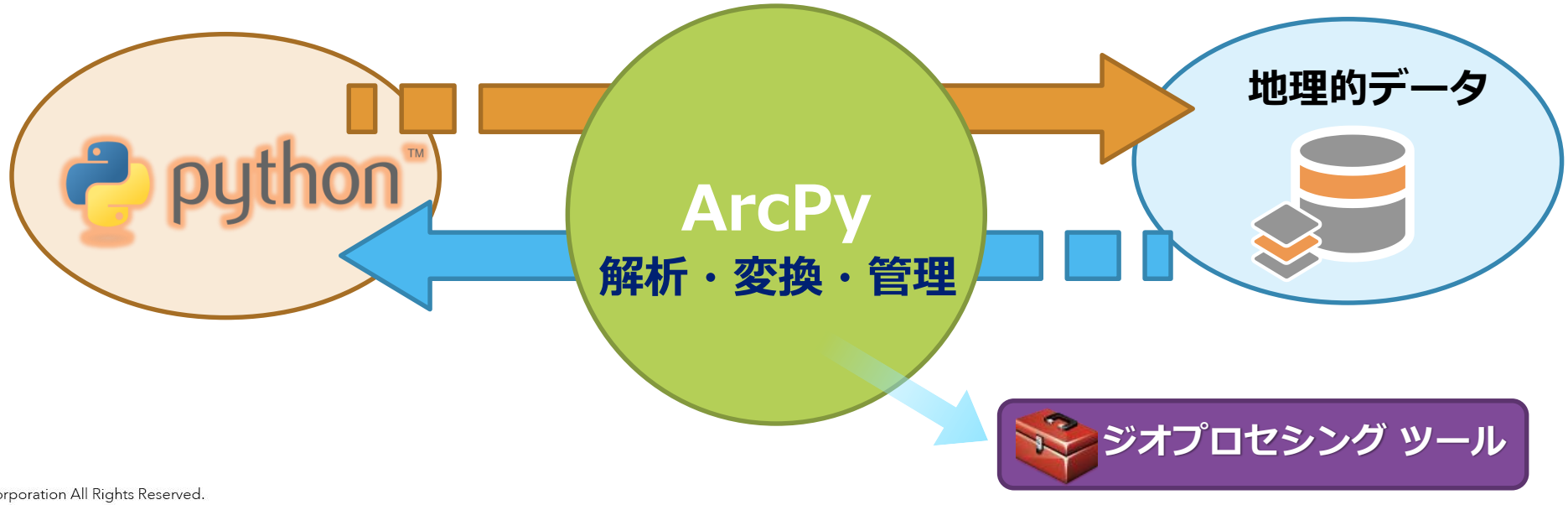
Python.exeの
プロセス内で処理できる

ArcPy とは

ArcPy とは

ArcGIS 製品群 のさまざまなタスクを自動化

- Python スクリプトから地理的データの解析、変換、管理などを実行するための便利な関数およびクラスの集まり（パッケージ）
- ジオプロセシング ツールを拡張するための標準ライブラリ
  - ArcGIS Desktop（ ArcMap / Pro ） / Engine / Enterprise
  - ArcGIS Runtime ローカルサーバー



ArcPy とは

Python/ArcPy を使うとできること

- **ArcGIS 上での一連の処理をスクリプト化**
  - プロパティの設定
  - ジオプロセシング ツールの実行
  - マップ操作

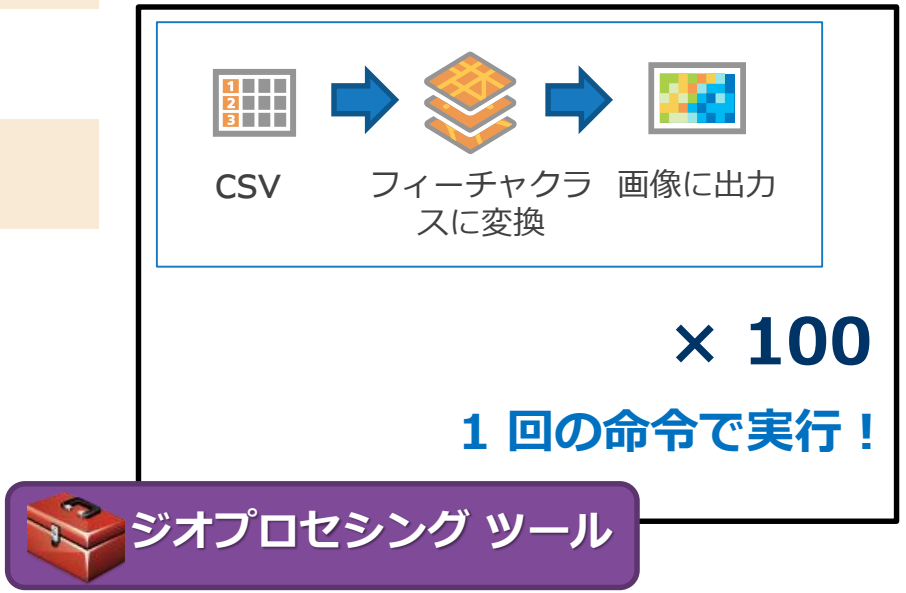
- **条件分岐を含む処理**
  - if（もし～なら）

```
if format == “csv”: #ファイル形式が CSV なら
    処理 1 を実行

if format == “shp”: #ファイル形式がシェープファイルなら
    処理 2 を実行
```

- **繰り返し (反復) 処理**
  - for (～条件範囲で繰り返す)

```
for layer in layers: #レイヤーの数分くりかえす
    繰り返し処理を実行
```

- **ジオプロセシング ツール化し汎用化**
  - 一連の処理を ArcPy で記述しツールとして登録
  - 作成したツールの保存・再配布も可能



ArcPy 関数

- 全てのジオプロセシング ツールが関数として提供されている
- ジオプロセシング のワークフローのサポートを後押しする関数も提供

ArcPy 関数一覧（アルファベット順）：
https://pro.arcgis.com/ja/pro-app/arcpy/functions/alphabetical-list-of-arcpy-functions.htm

例）

```
import arcpy

input = arcpy.GetParameterAsText(0)
if arcpy.Exists(input):
    print("Data exists")
else:
    print("Data does not exist")
```

ArcPy 関数の利用：
https://pro.arcgis.com/ja/pro-app/arcpy/geoprocessing_and_python/using-functions-in-python.htm

ArcPy クラス

- **ArcPy サイト パッケージで定義されたクラス**

ArcPy クラス一覧（アルファベット順）：
https://pro.arcgis.com/ja/pro-app/arcpy/classes/alphabetical-list-of-arcpy-classes.htm

例）

```
import arcpy

# 空間参照クラスを Web メルカトル（3857）で設定
spref = arcpy.SpatialReference(3857)

# SpatialReference の name と type 出力
print(spref.name)
print(spref.type)
```

```
import arcpy

# Pointクラスで、PointA と B のインスタンスをそれぞれ作成
pointA = arcpy.Point(2.0, 4.5)
pointB = arcpy.Point(3.0, 7.0)
```

ArcPy クラスの利用：
https://pro.arcgis.com/ja/pro-app/arcpy/geoprocessing_and_python/how-to-use-classes-in-python.htm

ArcPy マッピング モジュール

- マップ ドキュメント/レイヤーの操作を行うためのモジュール
- エクスポートと印刷を自動化する機能を提供

クラス/メソッド（本セッションで使うもの）	機能
ArcGISProject	プロジェクト ファイル（.aprx）のプロパティとメソッドへのアクセスを提供
listMaps	単一のプロジェクト ファイル（.aprx）内に存在するマップ オブジェクトの Python リストを返します。
listLayers	プロジェクト ファイル内のマップ、またはレイヤー ファイル（.lyr）内のレイヤーに存在するレイヤー オブジェクトの Python リストを返します。
exportToPDF	プロジェクト ファイル（.aprx）のページ レイアウトを PDF にエクスポートします。

例）

```
aprx = arcpy.mp.ArcGISProject(r"C:/Project/Watersheds.aprx")
aprx.listLayouts()[0].ExportToPDF(r"C:/Project/Output/Watersheds.pdf")
```

arcpy.mp モジュール 入門：
https://pro.arcgis.com/ja/pro-app/arcpy/mapping/introduction-to-arcpy-mp.htm

ArcPy データ アクセス モジュール

- データを扱うためのモジュール
- フィーチャ クラス や テーブル の レコード にアクセスして操作

クラス（本セッションで使うもの）	機能
InsertCursor	フィーチャ クラスまたはテーブルに書き込みカーソルを設定
SearchCursor	フィーチャ クラスまたはテーブルから返されたレコードへの読み取り専用アクセス
UpdateCursor	フィーチャ クラスまたはテーブルから返されたレコードへの読み書きアクセス

例）

```
import arcpy

fc = 'c:/data/base.gdb/well'
fields = ['WELL_ID', 'WELL_TYPE', 'SHAPE@XY']

# それぞれのレコードの WELL_ID , WELL_TYPE フィールド値、
# フィーチャの X , Y 座標を出力
with arcpy.da.SearchCursor(fc, fields) as cursor:
    for row in cursor:
        print(u'{0}, {1}, {2}'.format(row[0], row[1], row[2]))
```

データ アクセス モジュール とは？：
https://pro.arcgis.com/ja/pro-app/arcpy/data-access/what-is-the-data-access-module-.htm

ArcPy の使い方

ArcPy サイト パッケージのインポート

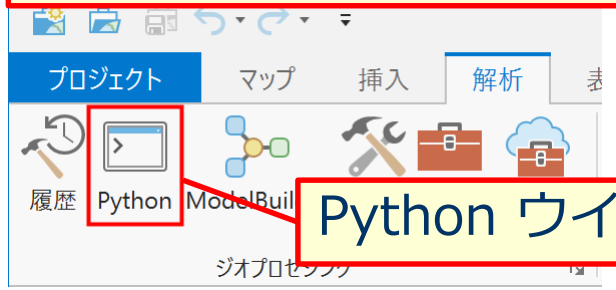
ArcPy の使い方

- ArcPy のサイト パッケージ をインポートする
- （Python ウィンドウを利用する場合は インポート不要）

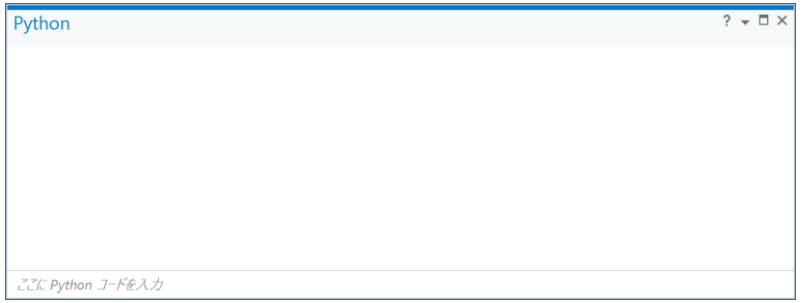
例）

```
# arcpy のインポート
import arcpy

# ArcGIS Pro で現在開いているプロジェクトファイルを取得
aprx = arcpy.mp.ArcGISProject("CURRENT")

# マップとレイヤーー覧を取得
maps = aprx.listMaps()[0]
lyrs =maps.listLayers()

# レイヤーの一覧を出力
for lyr in lyrs:
    print(lyr.name)

＞＞ レイヤー名１
＞＞ レイヤー名２
```

import arcpy
により、ArcPy サイトパッケージが提供している関数、クラス、モジュールが使用できるようになる



Python ウインドウの起動

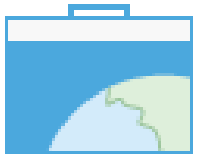


マップ・レイヤーの操作

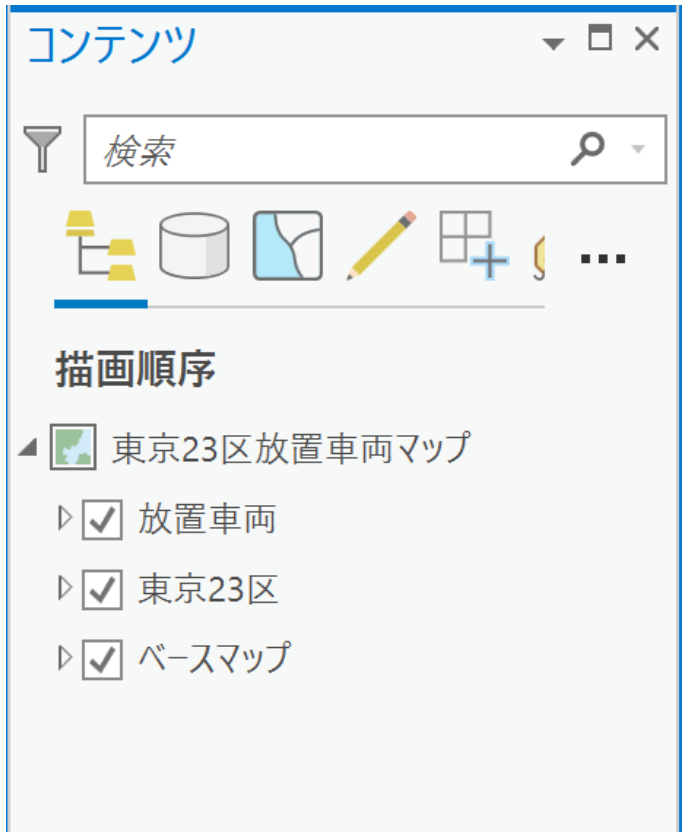
マッピングモジュール

マップ・レイヤーの操作

基本フロー



＝プロジェクト ファイル
（＊.aprx）



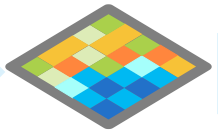
＝マップ[0]

＝レイヤー[0]

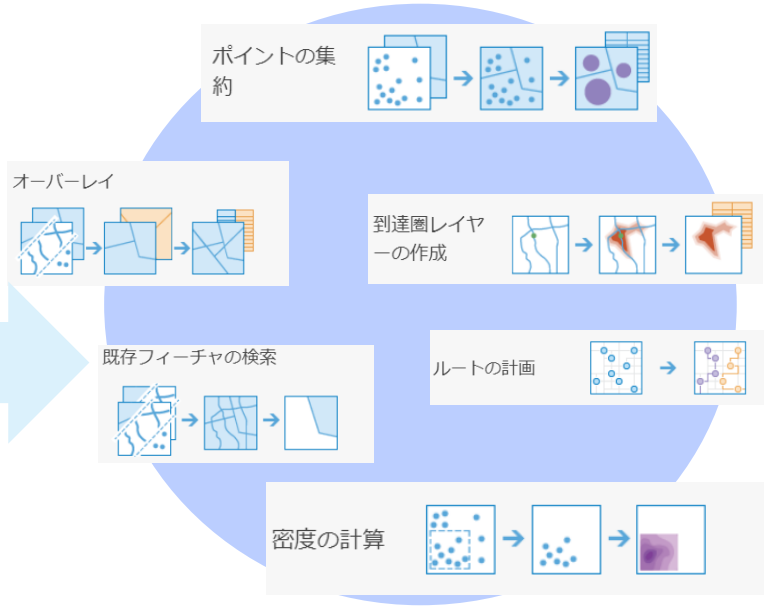
＝レイヤー[1]

＝レイヤー[2]

レイヤーのリスト



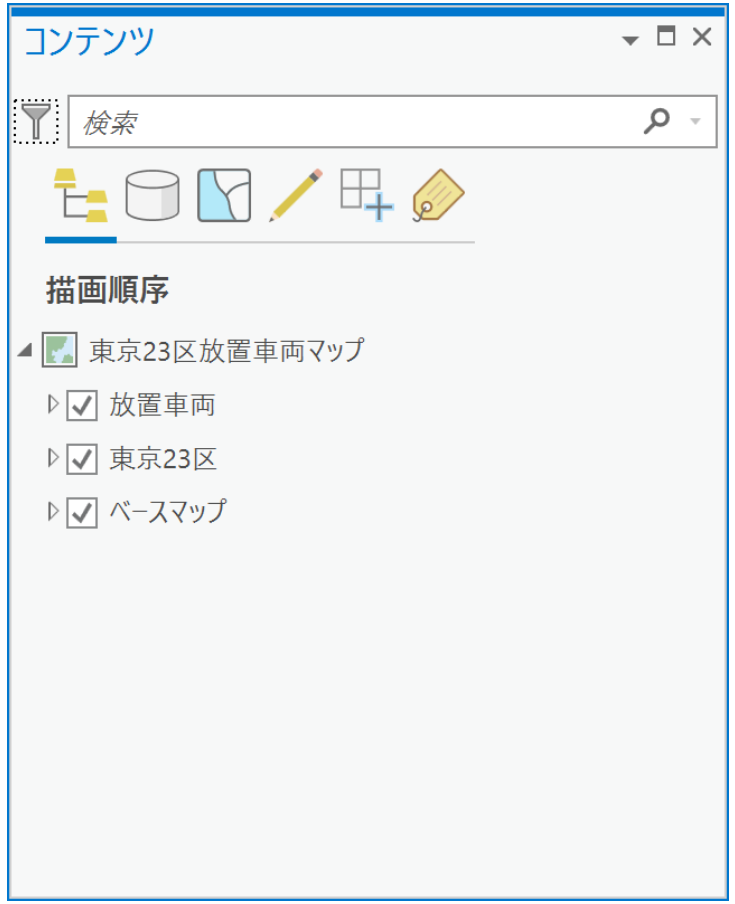
レイヤー
オブジェクト



解析処理や設定

プロジェクト ファイルの参照

ArcGISProject 関数



- **ArcGIS Pro で現在開いているプロジェクト ファイルを取得**

```
# “CURRENT” キーワードを使用
aprx = arcpy.mp.ArcGISProject(“CURRENT”)
```

マッピング モジュール　ArcGISProject 関数　プロジェクト ファイルのパス

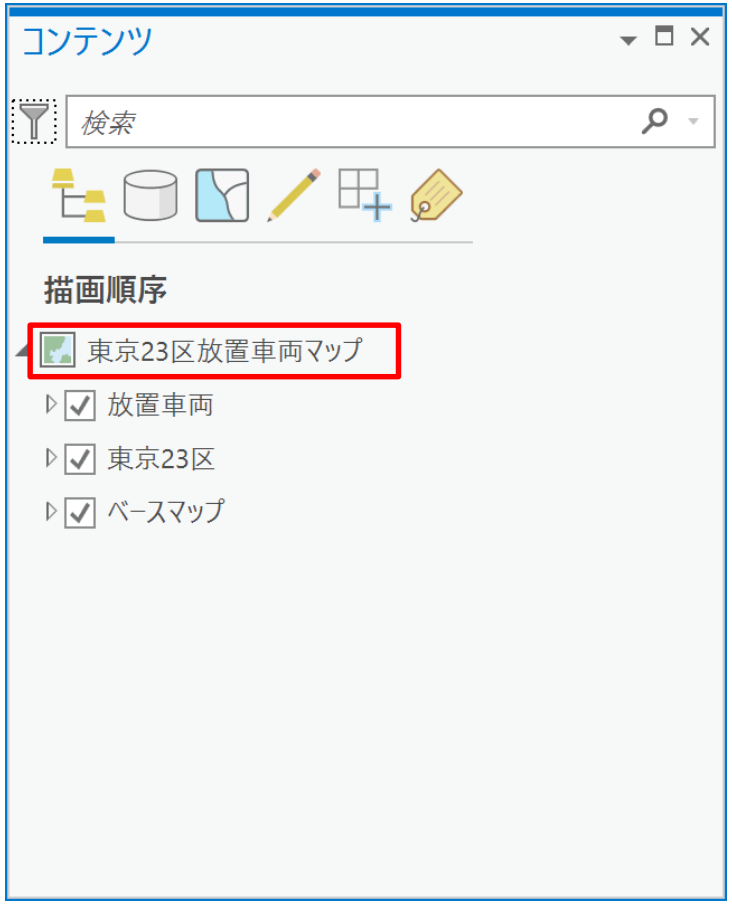
「CURRENT」 キーワードを使うと現在 ArcGIS Pro で
ロードされているプロジェクト ファイルを参照する

構文：ArcGISProject（aprx_path）

パラメータ	説明	データ型
aprx_path	プロジェクト ファイル（**.aprx**）のフルパス、 または CURRENT キーワードの文字列。	文字列

https://pro.arcgis.com/ja/pro-app/arcpy/mapping/arcgisproject-class.htm

マップの参照
listMaps メソッド



- **プロジェクト ファイルからマップの一覧リストを取得**

```
# プロジェクト ファイルのマップをリストとして取得
maps = aprx.listMaps(“<マップ名>”)
```

listMaps メソッド

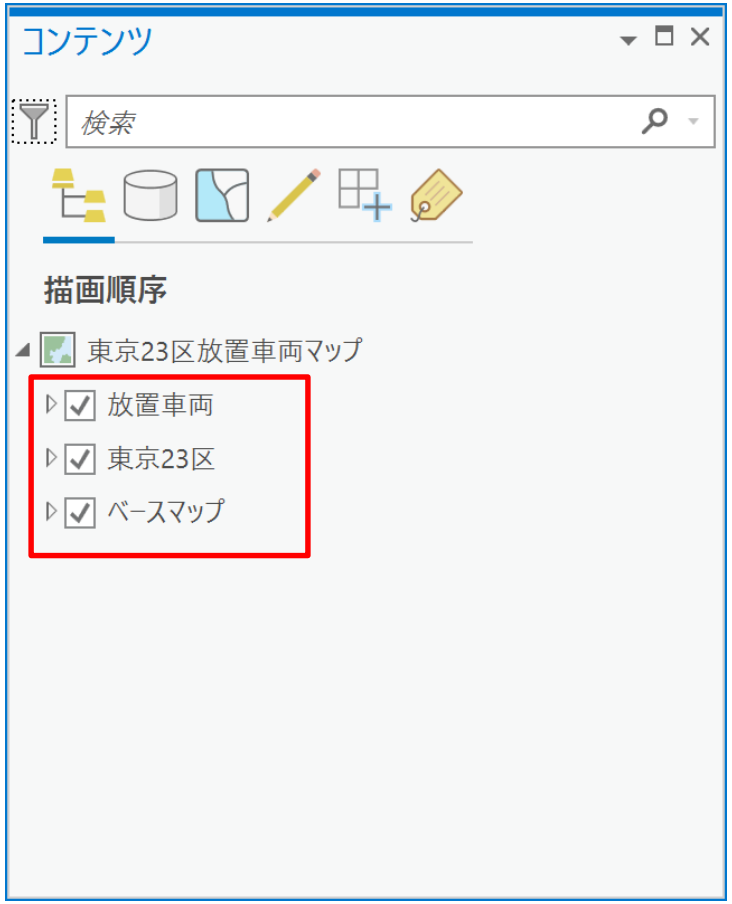
マップ オブジェクト

構文：listMaps（ {ワイルドカード}）

パラメータ	説明	データ型
ワイルドカード	アスタリスク（*）と文字の組み合わせを使用して指定した文字列が含まれるデータ フレームを取得する（デフォルトでは何も設定されていません）	文字列

https://pro.arcgis.com/ja/pro-app/arcpy/mapping/arcgisproject-class.htm

レイヤーの参照
listLayers メソッド



- **プロジェクト ファイル のマップに含まれるレイヤー リストを取得**

```
# マップに含まれるレイヤーの一覧を取得
lyrs =maps.listLayers()
```

マップ オブジェクト　　**listLayers メソッド**

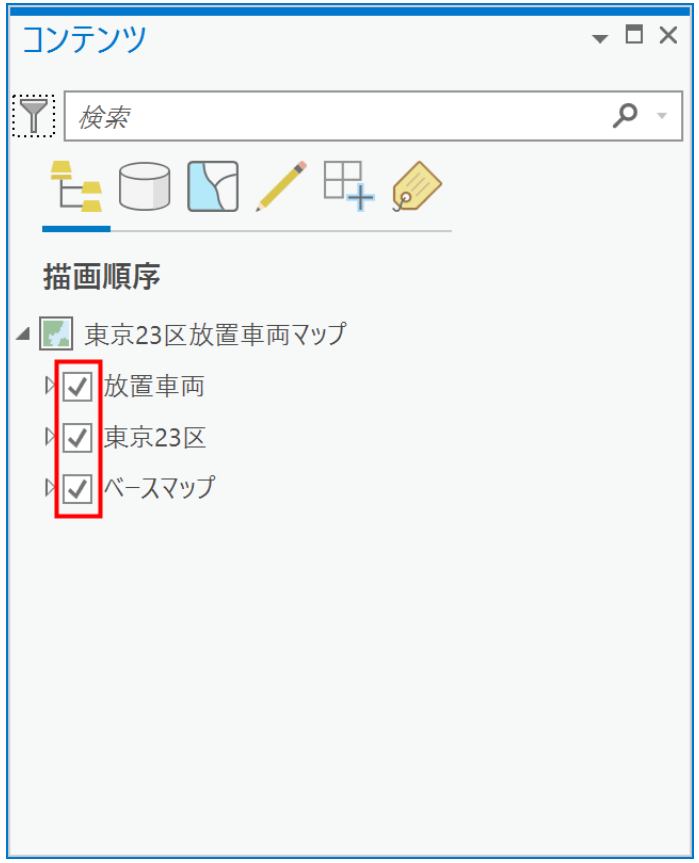
構文：listLayers（ {ワイルドカード}）

パラメータ	説明	データ型
ワイルドカード	アスタリスク（*）と文字の組み合わせを使用して指定した文字列が含まれるレイヤーを取得する（デフォルトでは何も設定されていません）	文字列

https://pro.arcgis.com/ja/pro-app/arcpy/mapping/layer-class.htm

表示の切り替え

表示・非表示の切り替え



- **レイヤーの表示・非表示**

```
# 非表示の場合は False
for lyr in lyrs:
    lyr.visible = True
```

レイヤー クラス が持つ visible プロパティに値をセット

構文： Layer (lyr_file_path) もしくは for layer in layers

パラメータ	説明	データタイプ
lyr_file_path	既存のレイヤー（.lyr）ファイルのフルパスとファイル名を含む文字列	文字列

プロパティ	説明	データタイプ
visible	レイヤの表示状態を制御します	ブール

https://pro.arcgis.com/ja/pro-app/arcpy/mapping/layer-class.htm

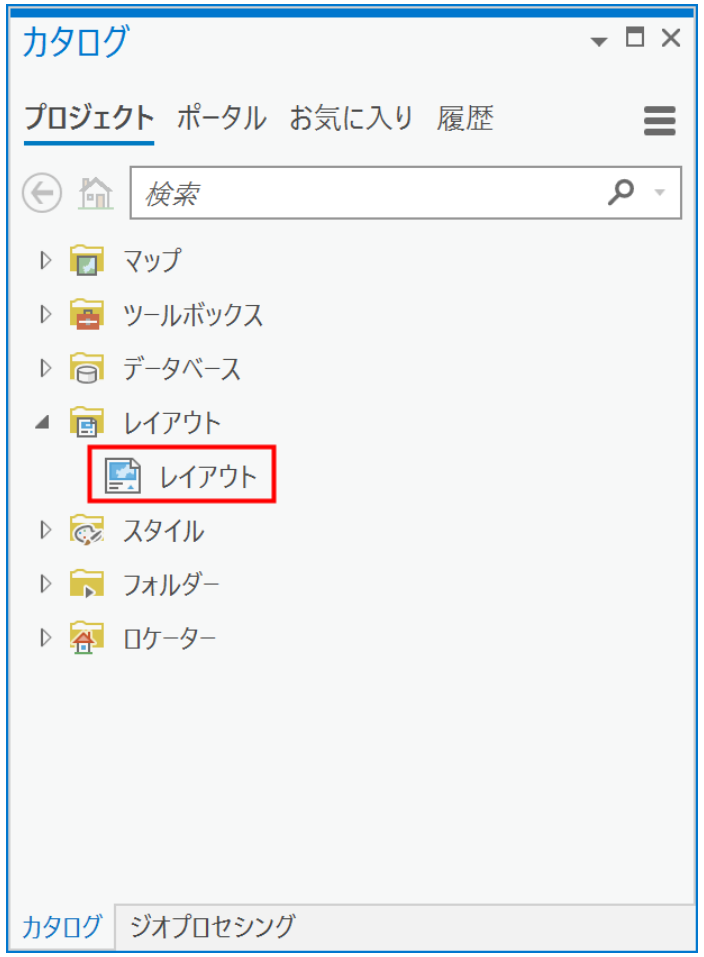
課題①：プロジェクト ファイルのオブジェクト操作

レイヤー オブジェクトを使ってレイヤーの表示非表示

```
# ArcPy サイト パッケージをインポートします。
import arcpy
# プロジェクト ファイルのオブジェクト取得
aprx = arcpy.mp.ArcGISProject("current")
# マップのオブジェクト取得
maps = aprx.listMaps()[0]
# レイヤーのオブジェクト取得
lyrs =maps.listLayers()

for lyr in lyrs:
    # レイヤーの表示設定
    lyr.visible = True
```

マップの出力

レイアウトの参照
listLayouts メソッド



- **プロジェクト ファイル に含まれるレイアウト リストを取得**

```
# プロジェクト ファイルに含まれるレイアウトの一覧を取得
aprx = arcpy.mp.ArcGISProject("CURRENT")
layout = aprx.listLayouts()[0]
```

マップ オブジェクト

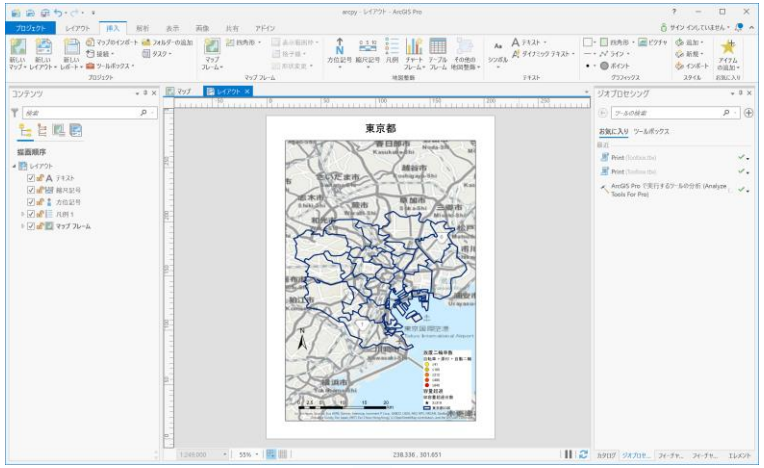
listLayouts メソッド

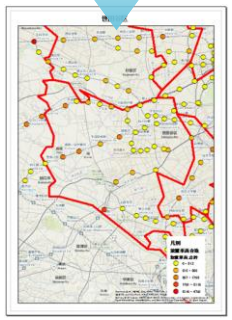
構文：listLayout（ {ワイルドカード}）

パラメータ	説明	データ型
ワイルドカード	アスタリスク（*）と文字の組み合わせを使用して指定した文字列が含まれるレイヤーを取得する（デフォルトでは何も設定されていません）	文字列

https://pro.arcgis.com/ja/pro-app/arcpy/mapping/arcgisproject-class.htm

マップの出力
exportToPDF メソッド





- **PDF へエクスポート**

```
#  すべてのオプションのデフォルト値を使用してページ レイアウトをPDF出力
aprx = arcpy.mp.ArcGISProject("CURRENT")
layout = aprx.listLayouts()[0]
layout.exportToPDF(r"C:¥data¥output¥Sample.pdf")
```

exportToPDF メソッド

PDF ファイルの出力先

構文：exportToPDF (out_pdf, {resolution}, {image_quality}, {compress_vector_graphics}, {image_compression}, {embed_fonts}, {layers_attributes}, {georef_info}, {jpeg_compression_quality}, {clip_to_elements}, {output_as_image})

パラメータ	説明	データタイプ
out_pdf	出力エクスポート ファイルのパスとファイル名を表す文字列	文字列

https://pro.arcgis.com/ja/pro-app/arcpy/mapping/mapview-class.htm

課題②：マップの操作

マップをファイルへ出力

```
# ArcPy サイト パッケージをインポートします。
import arcpy

# プロジェクト ファイルのオブジェクト取得
aprx = arcpy.mp.ArcGISProject("CURRENT")

# レイアウト[0]を取得
layout = aprx.listLayouts()[0]

#  ページ レイアウト設定内容を PDF へ出力
layout.exportToPDF(r"C:¥data¥output¥Sample.pdf")
```

```
#  ページ レイアウト設定内容をJPEG へ出力
layout.exportToJPEG(r"C:¥data¥output¥Sample.jpg")
```

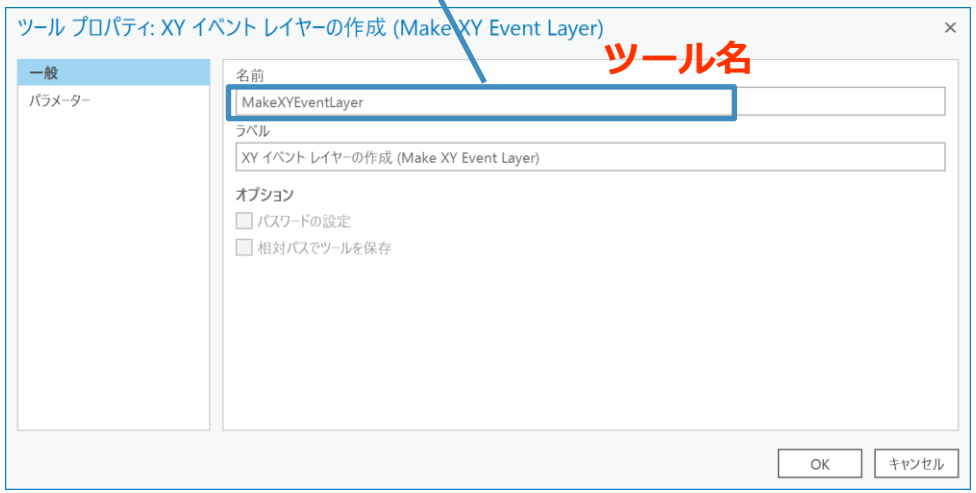
ジオプロセシング ツールの操作

ジオプロセシング ツールの使用

- **ArcPy からジオプロセシング ツール実行可能**

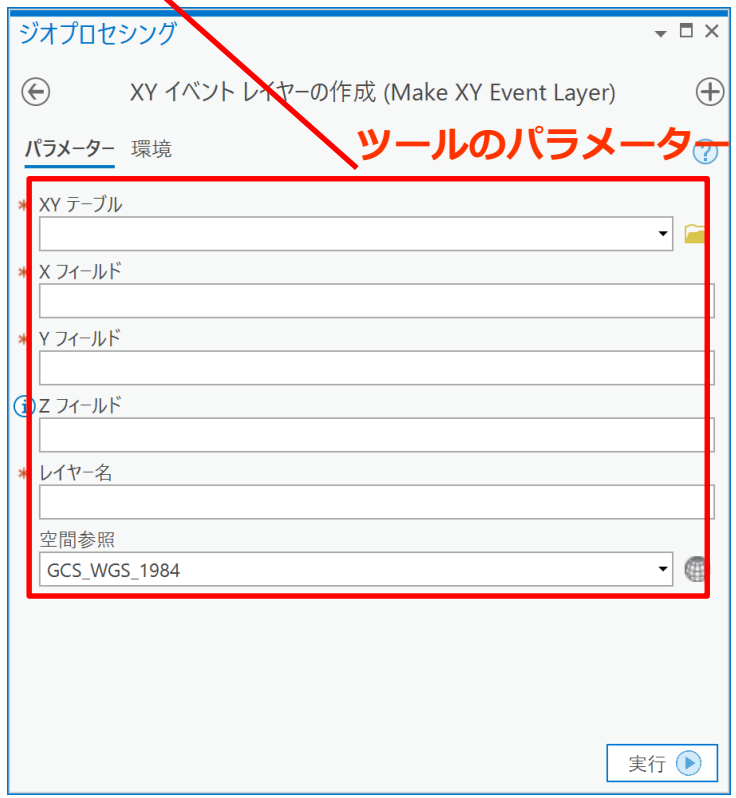
```
# ジオプロセシング ツールの基本的な呼び出し方
arcpy.MakeXYEventLayer_management( table, x_field , y_field , out_layer )
```

＊オプションは省略可



ツール名

ツールボックスの
エイリアス名



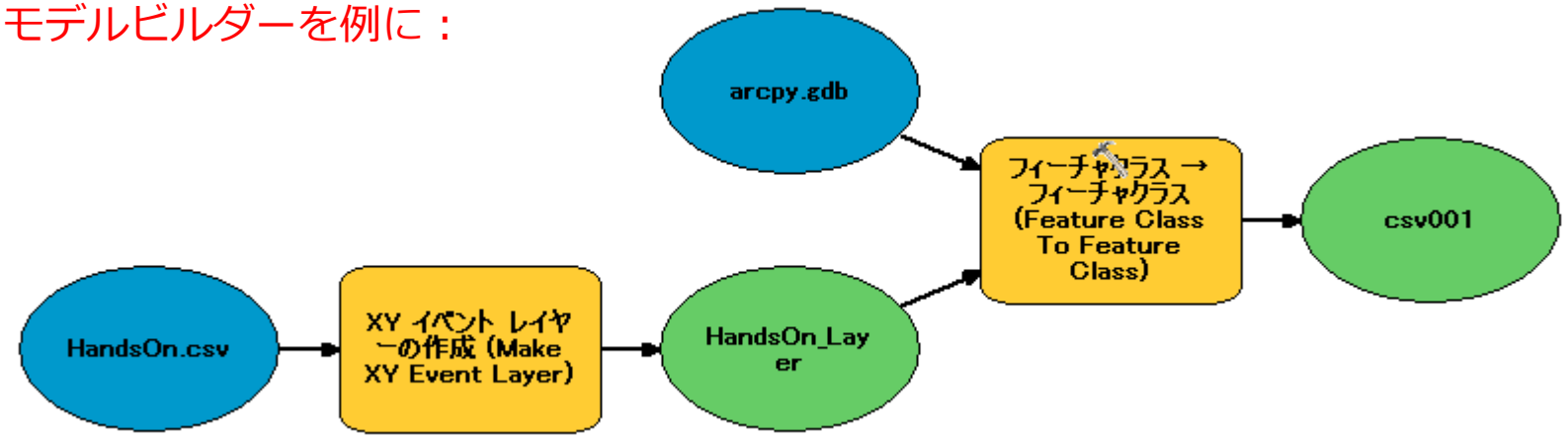
ツールのパラメーター

- [データ管理ツール] > [レイヤーとテーブル ビュー] > [XY イベント レイヤーの作成] ツールの例

ジオプロセシング ツールの使用

- ツールの出力結果を別のツールの入力値として使用する

モデルビルダーを例に：




```
# XY イベント レイヤー ツールを実行して CSV から HandsOn_layer を作成
arcpy.MakeXYEventLayer_management( table, x_field , y_field , HandsOn_layer )

# HandsOn_layer を フィーチャクラス として保存します。
arcpy.FeatureClassToFeatureClass_conversion(HandsOn_layer, out_path, “csv001”)
```

課題③：ジオプロセシング ツールの実行

ジオプロセシング ツール を Python から呼び出して実行

```
# ArcPy サイト パッケージをインポートします。
import arcpy

# ジオプロセシング ツールで利用する変数を定義
csv = r“C:¥data¥HandsOn.csv“
x = ”X”
y = ”Y“
layer = “PointLayer”
outpath = r”C:¥data¥ArcGISPro¥arcpy.gdb“

# CSV ファイルから XY イベント レイヤー（ テンポラリ のオブジェクト ）を作成します。
arcpy.MakeXYEventLayer_management(csv, x, y, layer)

# XY イベント レイヤー を フィーチャ クラス として保存します。
arcpy.FeatureClassToFeatureClass_conversion(layer, outpath, “csv001”)

# XY イベント レイヤー を削除します。
arcpy.Delete_management(layer)
```

データの選択

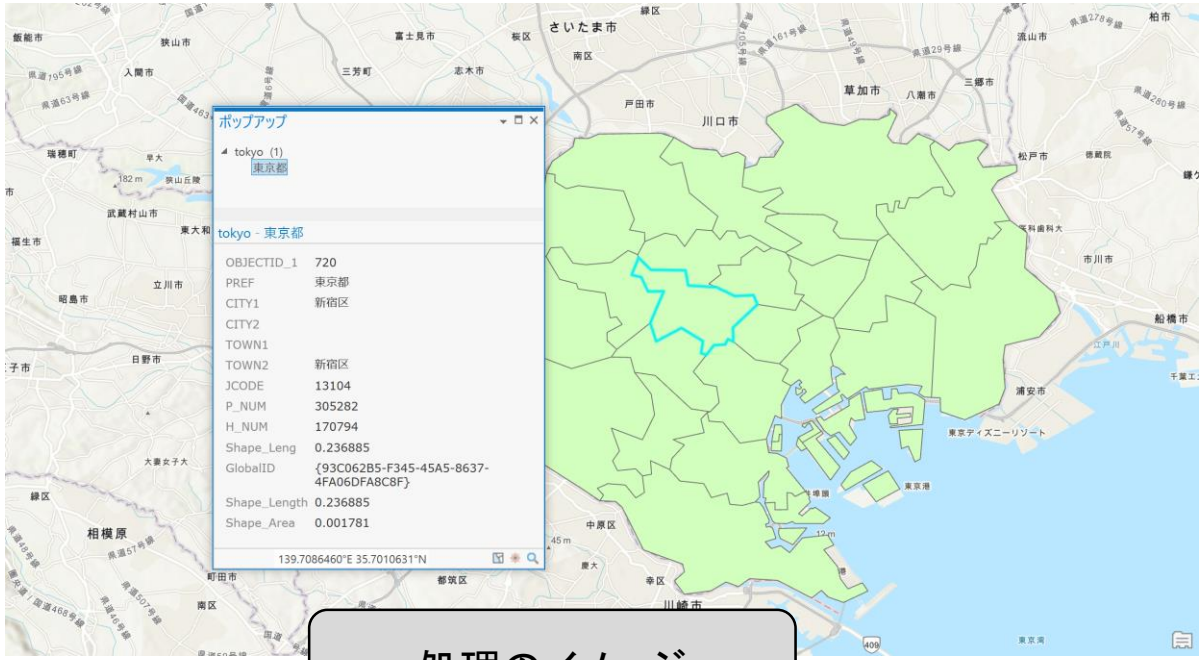
属性検索
空間検索

Select Layer By Attribute (属性検索)

- 属性検索に基づいて、レイヤーでの選択を追加、更新、削除

```
arcpy.SelectLayerByAttribute_management ("tokyo", "NEW_SELECTION", "CITY1 = '新宿区'")
```

パラメータ3　パラメータ1　パラメータ2



処理のイメージ

	パラメータ	説明
1	in_layer_or_view	選択するフィーチャ レイヤーまたはテーブル ビュー
2	selection_type (オプション)	どのように選択を適用する ・NEW_SELECTION - 結果として得られる選択によって既存の選択が置換されます ・ADD_TO_SELECTION - 現在選択しているものに加えて結果として得られる選択加わる ・REMOVE_FROM_SELECTION ・SUBSET_SELECTION ・SWITCH_SELECTION ・CLEAR_SELECTION
3	where_clause (オプション)	レコードのサブセットを選択するために使用する SQL 式

https://pro.arcgis.com/ja/pro-app/tool-reference/data-management/select-layer-by-attribute.htm

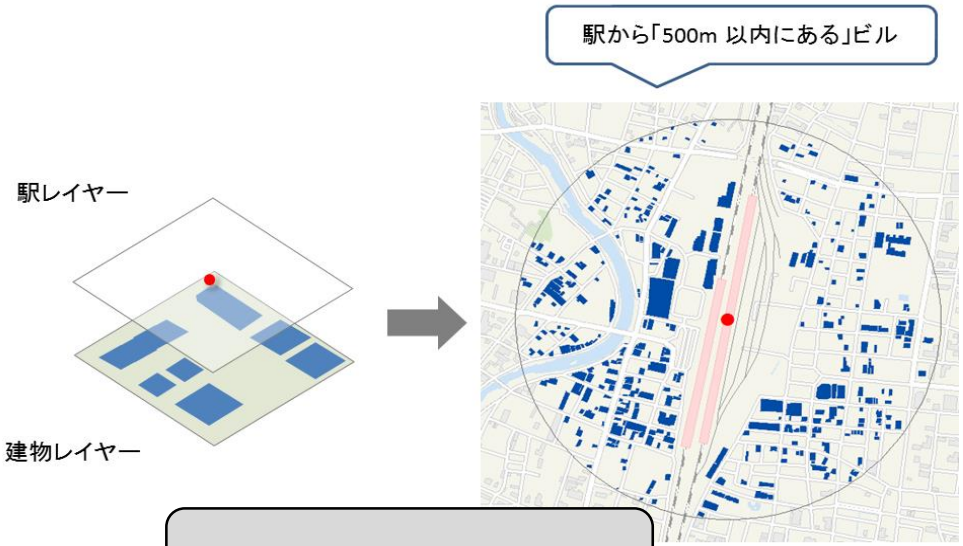
Select Layer By Location (空間検索)

- 空間的な関係性に基づいて、レイヤーでの選択を追加、更新、削除

パラメータ1

```
arcpy.SelectLayerByLocation_management("building",
"WITHIN_A_DISTANCE","ChibaStation","500 meters","NEW_SELECTION")
```

パラメータ2　パラメータ3　パラメータ4　パラメータ5



処理のイメージ

	パラメータ	説明
1	in_layer	select_features に対して評価されるフィーチャを含むレイヤー
2	overlap_type (オプション)	評価される空間リレーションシップ ・INTERSECT ・INTERSECT_3D ・WITHIN_A_DISTANCE ・ARE_IDENTICAL_TO など
3	select_features (オプション)	評価するフィーチャ クラス
4	search_distance (オプション)	Overlap_type によって選択可能
5	selection_type (オプション)	どのように選択を適用する ・NEW_SELECTION など
6	invert_spatial_relationship(オプション)	NOT_INVERT INVERT

https://pro.arcgis.com/ja/pro-app/tool-reference/data-management/select-layer-by-location.htm

リスト関数と Describe 関数

リスト関数

- **ArcPy には、フィールド、インデックス、データ セット、フィーチャ クラス、ファイル、ラスタ、テーブルなどのリストを取得する関数が用意されています。**

クラス（一部）	機能
ListFields	入力値に含まれるフィールドのリストを返します
ListFeatureClasses	現在のワーク スペース内にあるフィーチャ クラスを返します
ListRasters	現在のワーク スペース内にあるラスターのリストを返します

例）

```
import arcpy
# ワーク スペースを設定
arcpy.env.workspace = r"C:/data/ArcGISPro/arcpy.gdb"

# fcList 変数に ListFeatureClasses() 関数の取得結果を代入
fcList = arcpy.ListFeatureClasses()

# フィーチャ クラスの一覧を表示
for fc in fcList:
    print(fc)
```

データのリスト作成：
https://pro.arcgis.com/ja/pro-app/arcpy/get-started/listing-data.htm

Describe 関数
オブジェクトの取得

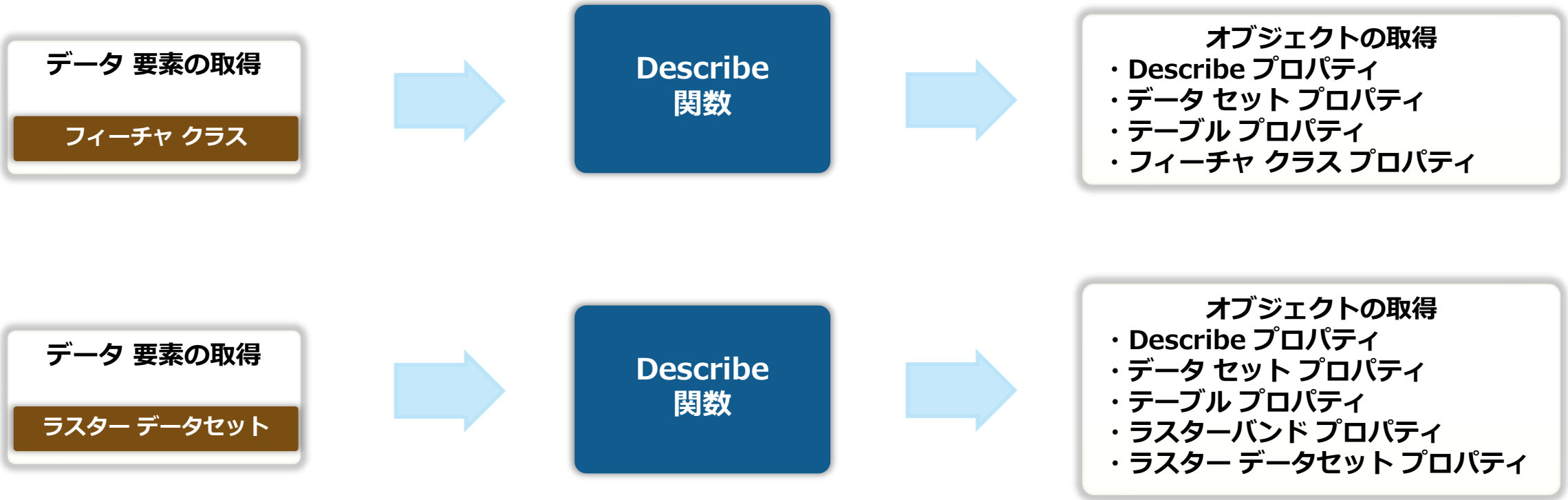
- 引数に指定したデータに応じてそれぞれのオブジェクトを取得

```
# Describe 関数にデータ要素を渡してデータ オブジェクトを取得
desc = arcpy.Describe("オブジェクト名")
```

Describe 関数

プロパティ情報

- Describe で取得したオブジェクトには様々なプロパティが含まれる
- 取得できる値をジオプロセシング処理へ利用可能



課題④：フィーチャのプロパティ の表示
データ取得

- フィーチャ クラスの情報を取得

```
import arcpy

# ワーク スペースを設定します。
arcpy.env.workspace = r"C:¥data¥ArcGISPro¥arcpy.gdb"

# fcList 変数に ListFeatureClasses() 関数の取得結果を代入
fcList = arcpy.ListFeatureClasses()

# フィーチャ クラスの一覧を表示
for fc in fcList:
    # 変数の代入（Describe オブジェクト）をします。
    desc = arcpy.Describe(fc)

    # Describe オブジェクトからフィーチャ クラスのプロパティを表示します。
    print(desc.name)
    print(desc.shapetype)
```

arcpy.da.cursor

データアクセスモジュール

arcpy.da.Cursor

カーソルとは

- カーソルは、表データに含まれるレコードの位置を指し示す。
- 検索、挿入、更新/削除することができるカーソルがある。

arcpy.da.Cursor

da.SearchCursor

SearchCursor を使用してフィーチャクラスからフィーチャを取得する

ID	shape	name	users
1	ポイント	永田町駅	10
2	ポイント	六本木駅	33
3	ポイント	赤坂見附駅	25

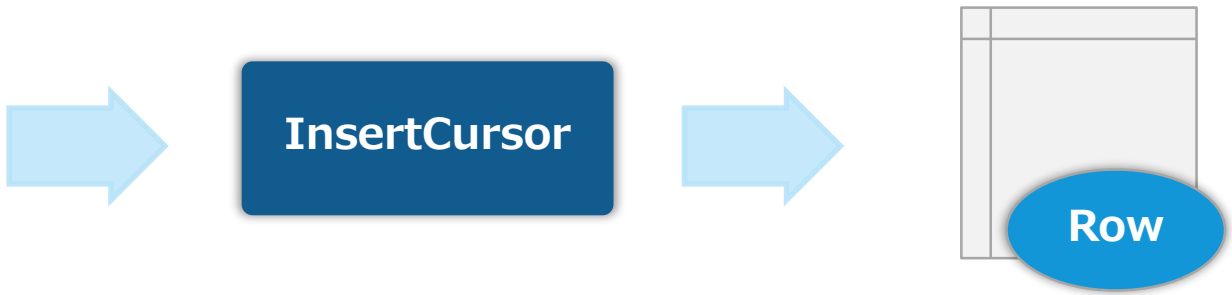

```
# SearchCursor クラスを使用して SearchCursor オブジェクトを取得します。
cur = arcpy.da.SearchCursor("Station", ["name", "users"])

# Cursor　から、name（ インデックス番号:0 ）フィールドと　users（ インデックス番号:1 ）フィールドの値を表示します。
for row in cur:
    print row[0]
    print row[1]

# オブジェクトを削除して、参照を解放します。
del cur

＞＞ 永田町駅
＞＞ 10
～中略～
＞＞ 25
```

arcpy.da.Cursor

InsertCursor

InsertCursor を使用してフィーチャ クラスにフィーチャを追加する

ID	shape	name	users
1	ポイント	永田町駅	10
2	ポイント	六本木駅	33
3	ポイント	赤坂見附駅	25




```
# 追加する値をタプル型で変数に格納します。
rowValues = ("表参道駅", (141.324167, 24.786667))

# InsertCursor オブジェクトを取得します。
cur = arcpy.da.InsertCursor("Station", ["name", "SHAPE@XY"])

# カーソル に 新しいレコード値をセットして 行を 追加します。
cur.insertRow(rowValues)

# オブジェクトを削除して、参照を解放します。
del cur
```

arcpy.da.Cursor

UpdateCursor

UpdateCursor を使用してフィーチャ クラスのフィーチャを更新する

ID	shape	name	users
1	ポイント	永田町駅	10
2	ポイント	六本木駅	33
3	ポイント	赤坂見附駅	25

ID	shape	name	users
1	ポイント	永田町駅	20
2	ポイント	六本木駅	33
3	ポイント	赤坂見附駅	25

```
# 更新対象の永田町駅 を UpdateCursor オブジェクトで取得します。
cur = arcpy.da.UpdateCursor("Station", "users", "name = '永田町駅' ")

# users ( インデックス番号:0 )フィールドの値を更新します。
for row in cur:
    row[0] = 20
    cur.updateRow(row)

# オブジェクトを削除して、参照を解放します。
del cur
```

課題⑤：カーソル

カーソルを使ってフィールドの値を更新

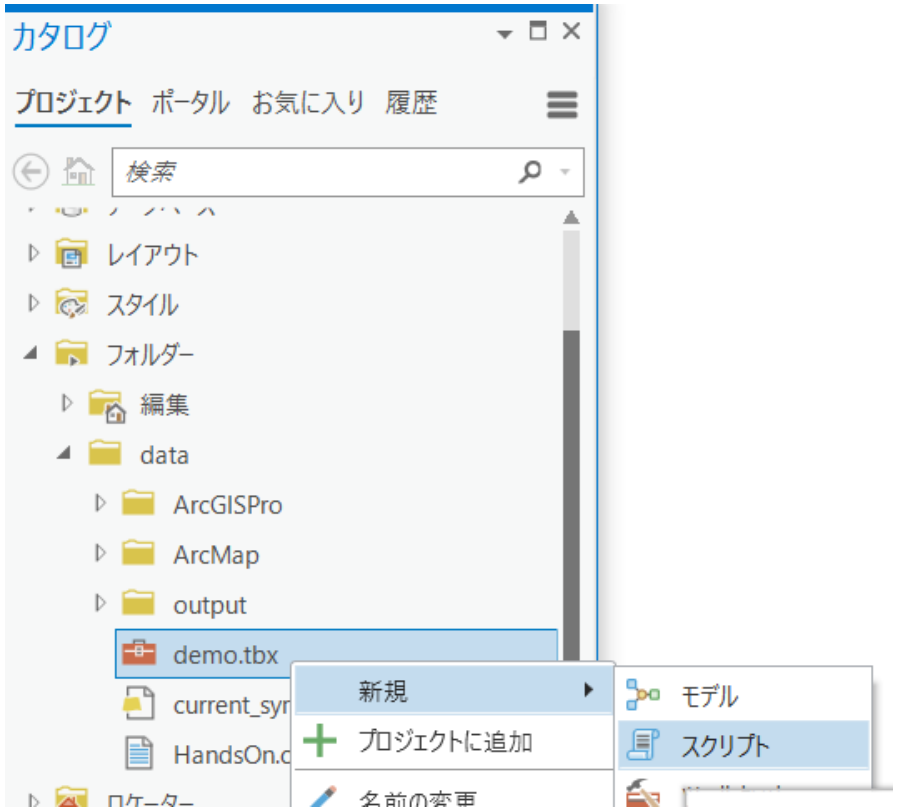
```
# ArcPy サイト パッケージをインポートします。
import arcpy

# csv001 フィーチャ クラス（課題③で作成）に LONG 型の total フィールドを追加します。
arcpy.AddField_management("csv001","total","LONG")

# csv001 フィーチャ クラスから取得するフィールド名のリストを作成
fields = ['total', 'bicycle', 'scooters', 'motorcycle']

# フィーチャ クラス (csv001) に対してカーソルを取得
cursor = arcpy.da.UpdateCursor("csv001", fields)

# for xx in :で、カーソルを移動しながら値を更新
for row in cursor:
    row[0] = row[1] + row[2] + row[3] # bicycle （ インデックス番号:1 ）～ motorcycle（ インデックス番号:3 ）の合計値
    cursor.updateRow(row) # 合計値の値を適用

# オブジェクトを削除して参照を解放します。
del cursor, row
```

スクリプト ツールの作成

スクリプト ツール

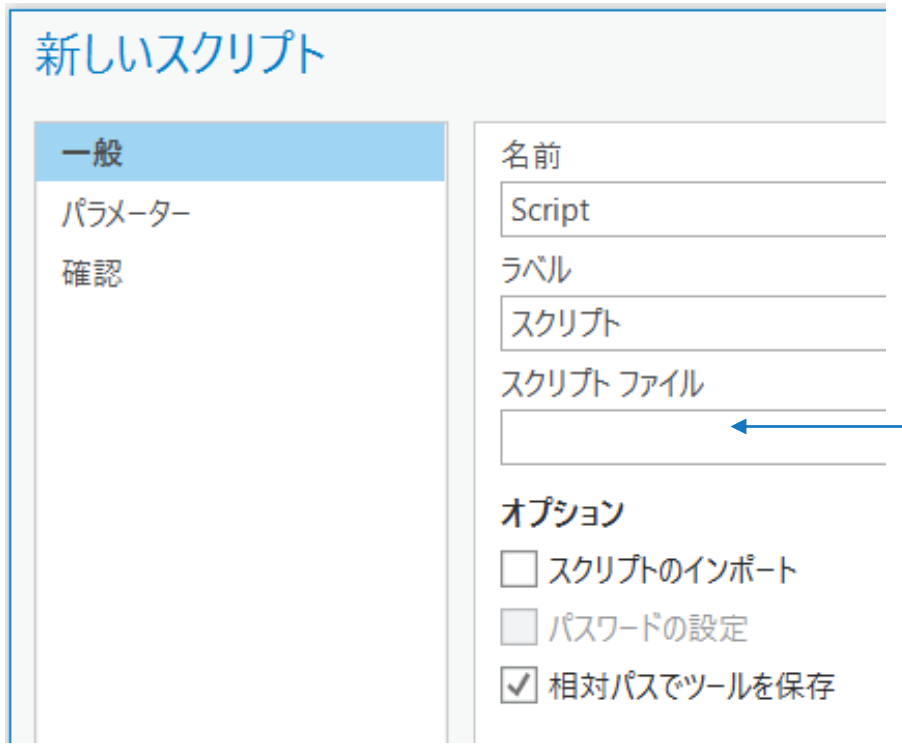
ArcToolbox 上に作成する Python ツール

- カスタム ツールボックス上に作成



スクリプト ツールの作成　～ 一般 ～

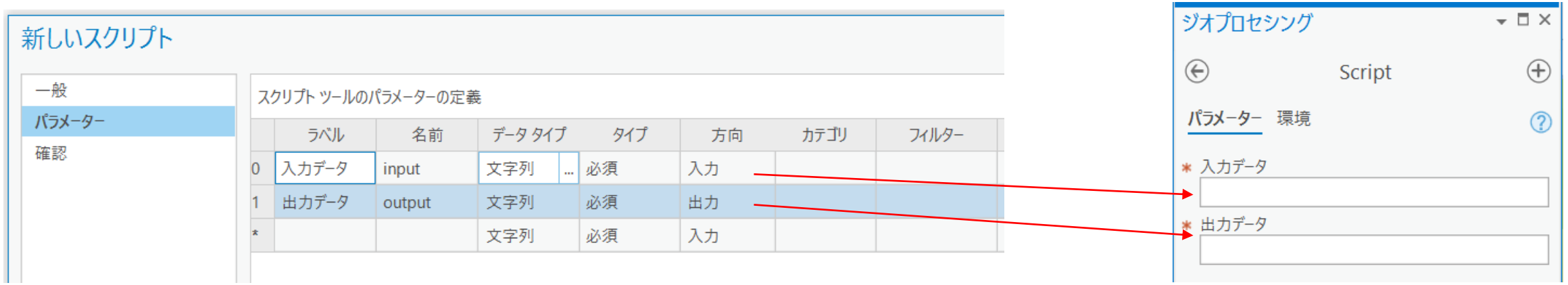
- 名前：ツールの名前（英数字のみ）
- ラベル：ツールの表示名
- スクリプト ファイル：ソース ファイル
- スクリプトのインポート
  - ツールボックスに
    ソース ファイルを組み込む
  - パスワード設定が可能に



スクリプト ツールの作成　～ パラメータ ～

- パラメータの定義
  - ラベル：パラメータの表示名
  - 名前：パラメータ名
  - データ タイプ：入力データタイプ
  - タイプ：必須 or オプション
  - 方向：入力 or 出力

```
import arcpy
arcpy.env.workspace =r"C:¥data¥ArcGISPro¥arcpy.gdb¥ChibaStation"

input = arcpy.GetParameterAsText(0)
output = arcpy.GetParameterAsText(1)

arcpy.Buffer_analysis(input, output, "500 meters")
```



パラメータを用意することでツール実行時にパラメータを指定できる

参考資料

ArcPy スタートアップガイド

- ArcMap と ArcGIS Pro での ArcPy を使用した演習
  - ArcGIS Desktop 製品ページ → [ドキュメント]
    - https://www.esrij.com/products/arcgis-desktop/documents/