Applied Numerical Methods for Civil Engineering

Week 3: Introduction to Python Programming: Part I

Xinyu Chen

Assistant Professor

University of Central Florida

How to understand

Applied Numerical Methods for Civil Engineering?

Numerical methods are techniques by which **mathematical problems** are formulated so that they can be solved with **arithmetic operations**.

Programming Environment

- **No prior programming experience required!**
- Setting up your **environment**
  - Free, no installation
  - Cloud-based Jupyter notebooks
  - Access anywhere with browser
  - Link: https://colab.research.google.com

Programming Environment

- **No prior programming experience required!**
- Setting up your **environment**
  - Free, no installation
  - Cloud-based Jupyter notebooks
  - Access anywhere with browser
  - Link: https://colab.research.google.com
- Try it now!

```
1 print('Hello Civil Engineering!')
2 print('Welcome to Applied Numerical Methods')
```

- What is print()?
  - A **function** that displays text
  - Anything in quotes is text (string)

Quizzes Now!

- **Today's participation** (ungraded survey): Please check out

  "Class Participation Quiz 5"

  Time slot: **2:30PM – 3:00PM**

  on Canvas.

- Online engagement (graded quizzes)

  "Quiz 5" (11 questions)

  Deadline: **11:59PM, January 26, 2026**

  on Canvas.

Variables: Storing Data

Variables are containers for data

```python
# Assign values to variables
length = 10.5      # meters
width = 5.2        # meters
material = 'Steel'
```

Rules for variable names:

1. Start with letter or underscore
2. Can contain letters, numbers, underscores
3. Case-sensitive: $Length \neq length$
4. Descriptive names recommended
5. Avoid Python keywords, e.g., `lambda`, `class`, `list`, `def`, etc.

Examples:

```python
length = 4
Length = 4.5
print('length = {}'.format(length))
print('Length = {}'.format(Length))
```

Basic Data Types

Four essential types

- Integers: Whole numbers $\ldots, -2, -1, 0, 1, 2, \ldots$

```
1  length = 4
```

- Floats: Decimal numbers

```
1  deflection = 0.025   # meters
```

- Strings: Text

```
1  material = 'Steel'
```

- Booleans: True/False

```
1  a = True
2  if a is True:
3      print(1)
4  else:
5      print(0)
```

Checking Data Types

- Use `type()` function:

```python
# Check types
length = 4
print(type(length))          # <class 'int'>

deflection = 0.025
print(type(deflection))          # <class 'float'>

material = 'Steel'
print(type(material))       # <class 'str'>

safe = True
print(type(safe))           # <class 'bool'>
```

- Why check types?
  - Different operations work with different types
  - Avoid errors like adding string to number
  - Understand what your code is doing

Basic Arithmetic Operations

Python programming example.

```
1 a = 2
2 b = 3
3 print(a + b)   # plus
4 print(a - b)   # minus
5 print(a * b)   # product
6 print(a / b)   # division
7 print(a ** 2) # quadratic function
8 print(a ** 3) # cubic function
```

Corresponding **arithmetic operations**:

Line **3**: $a + b$

Line **4**: $a - b$

Line **5**: $a \cdot b$

Line **6**: $\dfrac{a}{b}$

Line **7**: a^2

Line **8**: a^3

Note: `a ** n` refers to a to the power of n, or a^n.

Basic Arithmetic Operations

Engineering example.
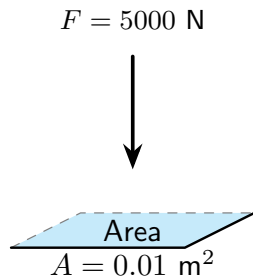
- Definition of normal stress:

$$\sigma = \frac{F}{A}$$

  where

  - $F = 5000$ N (force)
  - $A = 0.01$ m^2 (area)

```
1  force = 5000    # N
2  area = 0.01     # m^2
3  stress = force / area   # Pa
4  print('stress = {}'.format(stress))
```

$F = 5000$ N

Area

$A = 0.01$ m^2

Order of Operations

Python follows PEMDAS:

1. Parentheses

2. Exponents

3. Multiplication

4. Division

5. Addition

6. Subtraction

```
1 # Different results!
2 a1 = 10 + 5 * 2      # (5*2 first)
3 a2 = (10 + 5) * 2    # (parentheses first)
```

$$a_1 = 10 + 5 \times 2 \qquad a_2 = (10 + 5) \times 2$$

Order of Operations

Python follows PEMDAS:

1. Parentheses

2. Exponents

3. Multiplication

4. Division

5. Addition

6. Subtraction

Which one is correct?

$$c = \frac{w}{24 \cdot E \cdot I}$$

```
1 w = 10 ** 4            # uniform load
2 E = 2 * 10 ** 11       # modulus
3 I = 3.25 * 10 ** (-4)  # moment of inertia
4 c1 = w / 24 * E * I
5 c2 = w / (24 * E * I)
```

Lists: Storing Multiple Values

- Lists store collections of data

```python
1  # List of beam deflections (mm)
2  deflections = [12.3, 15.7, 18.2, 14.9, 16.5]
3  print(deflections)  # [12.3, 15.7, 18.2, 14.9, 16.5]
4
5  # List of materials
6  materials = ['Steel', 'Concrete', 'Timber', 'Aluminum']
7
8  # Access elements (0-indexed!)
9  print(deflections[0])   # First: 12.3
10 print(deflections[-1])  # Last: 16.5
```

Lists: Storing Multiple Values

- Lists store collections of data

```python
1  # List of beam deflections (mm)
2  deflections = [12.3, 15.7, 18.2, 14.9, 16.5]
3  print(deflections)  # [12.3, 15.7, 18.2, 14.9, 16.5]
4
5  # List of materials
6  materials = ['Steel', 'Concrete', 'Timber', 'Aluminum']
7
8  # Access elements (0-indexed!)
9  print(deflections[0])   # First: 12.3
10 print(deflections[-1])  # Last: 16.5
```

- List operations for engineering data

```python
1  print(len(deflections))        # Number of deflections
2  print(min(deflections))        # Minimum deflection
3  print(max(deflections))        # Maximum deflection
4  print(sum(deflection))         # Total
5  print(sum(deflections)/len(deflections))  # Average
```

Conditionals (if/elif/else)

- Make decisions in code:

```python
stress = 235   # MPa

if stress > 250:
    print('WARNING: Stress exceeds yield strength!')
elif stress > 200:
    print('Alert: Stress approaching limit')
else:
    print('Stress within safe limits')
```

Conditionals (if/elif/else)

- Make decisions in code:

```
1 stress = 235  # MPa
2
3 if stress > 250:
4     print('WARNING: Stress exceeds yield strength!')
5 elif stress > 200:
6     print('Alert: Stress approaching limit')
7 else:
8     print('Stress within safe limits')
```

- Comparison operators:
  - > greater than
  - < less than
  - >= greater or equal
  - <= less or equal
  - == equal to
  - != not equal to

Logical Operators (and/or/not)

- Use the logical operator and:

```
1  stress = 235
2
3  if stress <= 250 and stress > 200:
4      print('Alert!')
5  else:
6      print('Others')
```

- Use the logical operator or:

```
1  stress = 235
2
3  if stress > 250 or stress > 200:
4      print('At least alert!')
5  else:
6      print('Safe!')
```

for **Loop: Repeating Tasks**

- Process each item in a sequence:

```python
# List of beam deflections
deflections = [12.3, 15.7, 18.2, 14.9, 16.5]  # mm

# Check each beam
for d in deflections:
    if d > 15:
        print('Deflection exceeds limit')
    else:
        print('Deflection is OK')
```

- Common pattern: Process each item in experimental data

range() **Function for Numerical Loops**

- Generate sequences of numbers:

```python
# Count from 0 to 4
for i in range(5):
    print(i)

# With start and end
for i in range(2, 6):
    print(i)

# With step
for i in range(0, 10, 2):
    print(i)
```

range() **Function for Numerical Loops**

- Generate sequences of numbers:

```python
# Count from 0 to 4
for i in range(5):
    print(i)

# With start and end
for i in range(2, 6):
    print(i)

# With step
for i in range(0, 10, 2):
    print(i)
```

Line **2**-**3** Result: 0, 1, 2, 3, 4

Line **6**-**7** Result: 2, 3, 4, 5

Line **10**-**11** Result: 0, 2, 4, 6, 8

while **Loop: Repeat Until Condition**

- Repeat while condition is true:

```
1 a = [1, 2, 3, 4, 5, 6, 7, 8]
2 i = 0
3 while a[i] < 6:
4     print(a[i])
5     i = i + 1
```

Result: 0, 1, 2, 3, 4, 5

Functions: Reusable Code Blocks

- **Quadratic formula**. Given $ax^2 + bx + c = 0$ ($a \neq 0$), the quadratic formula is

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```python
1  import numpy as np
2
3  def quad_formula(a, b, c):
4      t = np.sqrt(b**2 - 4*a*c)
5      x1 = (-b + t) / (2*a)
6      x2 = (-b - t) / (2*a)
7      return x1, x2
```

Line **4** Compute $t = \sqrt{b^2 - 4ac}$

Line **5** Compute $x_1 = \dfrac{-b + t}{2a}$

Line **6** Compute $x_2 = \dfrac{-b - t}{2a}$

Functions: Reusable Code Blocks

- Given **parameters**: uniform load $w = 1 \times 10^4$ kg/m, modulus $E = 2 \times 10^{11}$ Pa, and moment of inertia $I = 3.25 \times 10^{-4}$ m^4.

- Compute the **constant factor**:

$$c = \frac{w}{24 \cdot E \cdot I} = \frac{10^4}{24 \times (2 \times 10^{11}) \times (3.25 \times 10^{-4})} = 6.41 \times 10^{-6}$$

```
1  import numpy as np
2
3  def const(w, E, I):
4      return w / (24 * E * I)
5
6  w = 10 ** 4              # uniform load
7  E = 2 * 10 ** 11         # modulus
8  I = 3.25 * 10 ** (-4)    # moment of inertia
9  c = const(w, E, I)       # constant factor
10 print(c)
```

Quick Summary

Monday's Class:

- Python environment (no installation with Colab)
- Introduction to Python: Variables, data types (integer, float, string, and Boolean).
- Arithmetic operations, order of operations.
- Storing multiple values with lists
- Logical operators (`for` and `while`)
- Defining functions by yourself

TBD