Applied Numerical Methods for Civil Engineering

CGN 3405 - 0002

Week 4: Introduction to Python Programming: Part II

Xinyu Chen

Assistant Professor

University of Central Florida

Quizzes Now!

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

  "`Class Participation Quiz 8`"

  Time slot: **2:30PM – 3:00PM**

  on Canvas.

Python Functions

Why use functions?

- **Reusability**: Write once, use many times
- **Modularity**: Break code into manageable blocks
- **Abstraction**: Hide complexity behind simple interfaces
- **Testing & Debugging**: Isolate and test individual components

Basic Function Syntax

```python
def function_name(parameters):
    """Optional docstring"""
    # Function body
    return value  # Optional
```

Basic Function Syntax

Engineering example.

- Definition of normal stress:

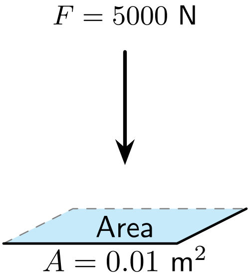
$$\sigma = \frac{F}{A}$$

```python
def normal_stress(F, A):
    return F / A
```

where

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

```python
force = 5000  # N
area = 0.01   # m^2
stress = normal_stress(force, area)
print('stress = {}'.format(stress))
```



Lambda Functions

Quick, one-line functions:

- Example: Quadratic function

$$y = x^2$$

```python
# Syntax: lambda arguments: expression
square = lambda x: x**2
print(square(5))      # 25

# Equivalent def function:
def square_func(x):
    return x**2
print(square_func(5)) # 25
```

Lambda Functions

Engineering example.

- Definition of normal stress:

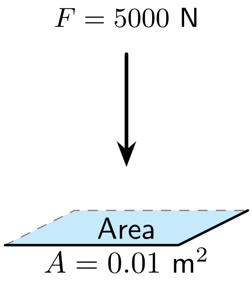
$$\sigma = \frac{F}{A}$$

```
stress_lam = lambda F, A: F / A
```

where

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

```
force = 5000  # N
area = 0.01   # m^2
stress = stress_lam(force, area)
print('stress = {}'.format(stress))
```



Lambda Functions

- Example:

$$g(r) = \frac{\pi r^2}{4}$$

```python
import numpy as np

g = lambda r: np.pi * x**2 / 4
```

- Evaluate it for $r = 1.5$ and $r = 2.78$

```python
print(g(1.5))
print(g(2.78))
```

Multiple Returns

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

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

```python
import numpy as np

def quad_formula(a, b, c):
    term = np.sqrt(b**2 - 4*a*c)
    x1 = (-b + term) / (2*a)
    x2 = (-b - term) / (2*a)
    return x1, x2
```

- Case study: Solve $9x^2 + 3x - 2 = (3x - 1)(3x + 2) = 0$.

```python
a, b, c = 9, 3, -2
x1, x2 = quad_formula(a, b, c)
print(x1)
print(x2)
```

Recursive Functions

Functions that call themselves

- **Factorial of a non-negative integer** n is the product of all positive integers less than or equal to n:

$$
\begin{aligned}
n! &= n \times (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1 \\
&= \begin{cases} 1 & \text{if } n = 1 \\ n \times \underbrace{(n-1)!}_{\text{factorial}} & \text{if } n > 1 \end{cases}
\end{aligned}
$$

```python
def factorial(n):
    f = 1
    for i in range(1, n + 1):
        f = f * i
    return f
```

- Toy example: $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

```python
print(factorial(5))
```

Recursive Functions

Functions that call themselves

- **Factorial of a non-negative integer** n is the product of all positive integers less than or equal to n:

$$
\begin{aligned}
n! &= n \times (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1 \\
&= \begin{cases} 1 & \text{if } n = 1 \\ n \times \underbrace{(n-1)!}_{\text{factorial}} & \text{if } n > 1 \end{cases}
\end{aligned}
$$

```python
def factorial_r(n):
    if n == 0:
        return 1
    else:
        return n * factorial_r(n-1)
```

- Toy example: $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

```python
print(factorial_r(5))
```

Factorial with NumPy

- **Factorial of a non-negative integer** n is the product of all positive integers less than or equal to n:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1$$

```python
def factorial_numpy(n):
    if n == 0:
        return 1
    else:
        return np.prod(np.arange(1, n+1))
```

- Toy example: $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

```python
print(factorial_numpy(5))
print(np.prod(np.arange(1, 6)))
```

Factorial with NumPy

- **Factorial of a non-negative integer** n is the product of all positive integers less than or equal to n:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1$$

```
def factorial_numpy(n):
    if n == 0:
        return 1
    else:
        return np.prod(np.arange(1, n+1))
```

- Toy example: $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

```
print(factorial_numpy(5))
print(np.prod(np.arange(1, 6)))
```

- Any other built-in function?

```
import math

print(math.factorial(5))
```

Approximation for Sine Function

Taylor series expansion for $\sin(x)$:

- Formula

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!} - \frac{x^{15}}{15!} + \cdots$$

- Denominator is factorial of odd numbers
- More terms = better approximation

Approximation for Sine Function

Taylor series expansion for $\sin(x)$:

- Formula

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!} - \frac{x^{15}}{15!} + \cdots$$

- Denominator is factorial of odd numbers
- More terms = better approximation
- Python programming:

$$\sin(x) = \underbrace{\sum_{n=1}^{+\infty}(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}}_{n \text{ starts from } 1}$$

$$= \underbrace{\sum_{n=0}^{+\infty}(-1)^{n}\frac{x^{2n+1}}{(2n+1)!}}_{n \text{ starts from } 0 \text{ (Python!)}}$$

Approximation for Sine Function

- Python programming:

$$\sin(x) = \underbrace{\sum_{n=0}^{+\infty}(-1)^n \frac{x^{2n+1}}{(2n+1)!}}_{n \text{ starts from } 0 \text{ (Python!)}}$$

```python
import numpy as np

def sin_taylor(x, num_term):
    result = 0
    for n in range(num_term):
        # Term index: 0, 1, 2, ... corresponds to x^1,
            x^3, x^5, ...
        exp = 2*n + 1
        factorial = np.prod(np.arange(1, exp + 1))
        result += ((-1) ** n) * (x ** exp) / factorial
    return result
```

Approximation for Sine Function

Test case: $\sin(0.9)$

- Ground-truth value:

```python
print(np.sin(0.9))        # 0.7833269096274834
```

- 1 term:

```python
print(sin_taylor(0.9, 1)) # 0.9
```

Approximation for Sine Function

Test case: $\sin(0.9)$

- Ground-truth value:

```
print(np.sin(0.9))          # 0.7833269096274834
```

- 1 term:

```
print(sin_taylor(0.9, 1)) # 0.9
```

- 2 terms:

```
print(sin_taylor(0.9, 2)) # 0.7785
```

- 3 terms:

```
print(sin_taylor(0.9, 3)) # 0.78342075
```

- 4 terms:

```
print(sin_taylor(0.9, 4)) # 0.7833258498214286
```

- 5 terms:

```
print(sin_taylor(0.9, 5)) # 0.7833269174484375
```

Quick Summary

Monday's Class:

- Basic function syntax
- Lambda function
- Multiple returns
- Recursive functions
- Two examples: Factorial and Taylor series expansion for $\sin(x)$