Functions

Aidan Hunt, University of Washington

Learning Objectives

After the lecture, students will be able to:

- Interpret Python function syntax
- Define Python functions that include input parameters and returns
- Document functions with a succint description of what it does, required inputs, and outputs
- Determine the information available inside and outside of a function via scope.

Check-in

- Python install going okay?

Motivation

- Functions are the building blocks of Python programs.
- Represent distinct tasks that can be called when needed

Built-in functions

- print function: print text to the console
- type function: get data type of a variable
- len function: get length of string or other sequence

In [49]:
```
# The print function for displaying messages or the contents of variables.
print("Hello there!")
```

```
Hello there!
```

In [50]:
```
# The type function for determining the type of data stored in a variable.
x = 5
type(x)
```

Out[50]:
```
int
```

In [51]:
```
# The length function for determining how many characters are in a string.
len('This is a sentence.')
```

Out[51]:
```
19
```

Functions like this are useful because they are performing common, useful actions on-demand.

Today we'll be going over

1. how to define our own functions

2. how to use functions to factor out redundancy

Defining basic functions

To define a function, use the `def` keyword, followed by indented lines of code.

For example, consider a simple function that calculates the Reynolds number for flow over a flat plate:

In [52]:
```
# Defining our function
def calcReynoldsNumber(U, L, nu):
    Re = U * L / nu
    return Re
```

Key points on syntax:

- The keyword `def` tells Python that we are declaring a function
- The function name comes after the `def` keyword.
- Parameters are how information is passed into the function. You can have as many inputs as you want (including 0).
- All lines of code *inside* the function are indented by one tab.
- Returns are how information is passed out of the function. You can have as many returns as you want (including 0).
- The function ends when 1) we indent back out or 2) the return keyword is encountered

Calling functions

In [53]:
```
# Calling our function
U = 5
L = 10
nu = 1e-6

Re = calcReynoldsNumber(U, L, nu)
print(Re)
```

```
50000000.0
```

Key points on calling functions:

- The names of parameters outside the function don't need to match the names of parameters inside the function.
    - Parameter "slots"
- Same for returns
    - Return "slots"
- You must "catch" a return to use it outside the function.

Questions?

- The syntax for defining functions is the easy part.
- Let's work an example to show how they can be useful for organization.

Example: Bernoulli calculation

Let's look at an example from fluid mechanics:

- Suppose there is fluid flowing in a pipe.
- We know the upstream pressure, velocity, and elevation.
- We know the downstream velocity and elevation, and would like to know the pressure.
- We can solve this with the Bernoulli equation.

In [54]:
```
# Insert photo here
```

In [55]:
```
# Constants
rho = 1000 # Density [kg/m^3]
g = 9.81 # Gravity [m/s^2]
Patm = 101.325 * 10**3 # Atmospheric pressure [Pa]
```

In [56]:
```
# Upstream properties
P1 = 10 * Patm
V1 = 3
z1 = 5
```

In [57]:
```
# Downstream properties
V2 = 5
z2 = 0
# P2 unknown
```

In [58]:
```
# Calculate the downstream pressure, P2 using the Bernoulli equation.
P2 = P1 + rho*g*z1 + 1/2*rho*V1**2 - rho*g*z2 - 1/2*rho*V2**2

print(P2)
```

```
1054300.0
```

Question What stands out in the calculation above?

- Repeated calculations for hydrostatic terms
- Repeated calculations for dynamic terms

In [59]:
```
# Let's define methods for these two repeated calculations

def calcHydrostaticPressure(z, rho, g):
    return z*rho*g

def calcDynamicPressure(V, rho):
    return 1/2*rho*V**2
```

In [60]:
```
# Now let's reduce redundancy by plugging them into our calculation with the appropriate

# Calculate the downstream pressure, P2
P2 = P1 + calcHydrostaticPressure(z1, rho, g) + calcDynamicPressure(V1, rho) \
        - calcHydrostaticPressure(z2, rho, g) - calcDynamicPressure(V2, rho)

print(P2)
```

```
1054300.0
```

In [61]:
```
# Finally, let's make this even more modular by making this calculation its own method
# Break the calculation into steps for readability

def calculateDownstreamPressure(P1, V1, V2, z1, z2, rho, g):
```

```python
    # Calculate difference between upstream and downstream hydrodynamic terms
    deltaHydrostatic = calcDynamicPressure(V1, rho) - calcDynamicPressure(V2, rho)

    # Calculate difference between upstream and downstream dynamic terms
    deltaDynamic = calcHydrostaticPressure(z1, rho, g) - calcHydrostaticPressure(z2, rho

    # Calculate downstream pressure and return
    P2 = P1 + deltaHydrostatic + deltaDynamic
    return P2
```

In [62]:
```python
# Call the function
P2 = calculateDownstreamPressure(P1, V1, V2, z1, z2, rho, g)
print(P2)
```

```
1054300.0
```

Advantages of this approach

- We can easily perform this calculation on several different cases and compare the results.
- Math errors can be fixed program-wide very easily.
- Our code is readable and organized into distinct steps.

How many functions should we define?

- We could, but eventually reach diminishing returns
  - Too many functions can make a program less readable.
  - If every line of code is a function - then you just have individual lines of code!
- Remember our goals for writing functions are to:
  - Factor out distinct tasks that we'd like to have access to on-demand
  - Make our program more readable and organized

What's missing from the functions we've defined?

Documentation! Every function we write should have a description that allows other people (or ourselves in the future) to understand what it does and how to use it.

Documenting functions

There's something important missing from this function: documentation.

- Tells someone unfamiliar with our program how to use it.
- Remind ourselves how to use our program.

Question: What information should we include in our documentation?

- A general description of what the function does. Don't need implementation details
- Describe each input parameters (what they mean, what types are expected, whether optional or not)
- Describe each return variable (what they mean, types expected)

What shouldn't go in our documentation?

- Implementation details (the nitty gritty)

Let's add these things as a "docstring" to our function.

- Multiline string
- Write below function definition

In [63]:
```python
# Option 1: Short docstring (good for simple functions)

def calcHydrostaticPressure(z, rho, g):
    '''
    Given an elevation (z), density (rho), and acceleration due to gravity (g),
    calculates and returns the hydrostatic pressure term from the
    Bernoulli equation.
    '''
    return z*rho*g

def calcDynamicPressure(V, rho):
    '''
    Given a velocity (V) and density (rho), calculates and returns the
    dynamic pressure term from the Bernoulli equation.
    '''
    return 1/2*rho*V**2
```

In [64]:
```python
# Option 2: List-type docstring (good for more complicated functions)
def calculateDownstreamPressure(P1, V1, V2, z1, z2, rho, g):
    '''
    Given known upstream and downstream fluid properties, as well as the density
    and acceleration due to gravity, calculates and returns the downstream pressure.

    Inputs:
        P1 - upstream pressure in Pa
        V1 - upstream velocity in m/s
        V2 - downstream velocity in m/s
        z1 - upstream elevation in m
        z2 - downstream elevation in m
        rho - fluid density in kg/m^3
        g - acceleration due to gravity in m/s^2

    Ouptuts
        The downstream pressure, P2, in Pa
    '''

    # Calculate difference between upstream and downstream hydrodynamic terms
    deltaHydrostatic = calcDynamicPressure(V1, rho) - calcDynamicPressure(V2, rho)

    # Calculate difference between upstream and downstream dynamic terms
    deltaDynamic = calcHydrostaticPressure(z1, rho, g) - calcHydrostaticPressure(z2, rho

    # Calculate downstream pressure and return
    P2 = P1 + deltaHydrostatic + deltaDynamic
    return P2
```

In [65]:
```python
# The documentation for our function is visible if we call the "help" function and provi
help(calculateDownstreamPressure)
```

```
Help on function calculateDownstreamPressure in module __main__:

calculateDownstreamPressure(P1, V1, V2, z1, z2, rho, g)
    Given known upstream and downstream fluid properties, as well as the density
    and acceleration due to gravity, calculates and returns the downstream pressure.

    Inputs:
        P1 - upstream pressure in Pa
        V1 - upstream velocity in m/s
        V2 - downstream velocity in m/s
```

```
        z1 - upstream elevation in m
        z2 - downstream elevation in m
        rho - fluid density in kg/m^3
        g - acceleration due to gravity in m/s^2

    Ouptuts
        The downstream pressure, P2, in Pa
```

In [66]:
```python
# You can do this for all functions.
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Questions?

Optional Parameters

Consider `g` in our function above.

- Passed as a parameter for completeness
- But the value will almost always be 9.81
- Specify default value in function definition!
- Can do the same for `rho`, as well.

Note

- In your function definition, optional inputs must be listed last.
- When passing variables into the optional inputs of a function, you can pass them in any order, as long as you use the proper keyword (e.g., `rho=850`) see below.

Make sure to update documentation accordingly!

In [67]:
```python
def calculateDownstreamPressure(P1, V1, V2, z1, z2, rho=1000, g=9.81):
    '''
    Given known upstream and downstream fluid properties, as well as the density
    and acceleration due to gravity, calculates and returns the downstream pressure.

    Required Inputs:
        P1 - upstream pressure in Pa
        V1 - upstream velocity in m/s
        V2 - downstream velocity in m/s
        z1 - upstream elevation in m
        z2 - downstream elevation in m

    Optional Inputs:
        rho - fluid density in kg/m^3 (default 1000)
        g - acceleration due to gravity in m/s^2 (default 9.81)
```

```
    Ouputs
        The downstream pressure, P2, in Pa
    '''

    # Calculate difference between upstream and downstream hydrodynamic terms
    deltaHydrostatic = calcDynamicPressure(V1, rho) - calcDynamicPressure(V2, rho)

    # Calculate difference between upstream and downstream dynamic terms
    deltaDynamic = calcHydrostaticPressure(z1, rho, g) - calcHydrostaticPressure(z2, rho

    # Calculate downstream pressure and return
    P2 = P1 + deltaHydrostatic + deltaDynamic
    return P2
```

In [68]:
```
# Now we can call it without rho and g (default values will be used)
P2 = calculateDownstreamPressure(P1, V1, V2, z1, z2)
print(P2)
```

```
1054300.0
```

In [69]:
```
# Calling with rho = 850 as optional input
P2 = calculateDownstreamPressure(P1, V1, V2, z1, z2, 850)
print(P2)
```

```
1048142.5
```

In [70]:
```
# Passing optional inputs out of order, using keywords from function definition
P2 = calculateDownstreamPressure(P1, V1, V2, z1, z2, g=100, rho=850)
print(P2)
```

```
1431450.0
```

Scope

The word "scope" refers to what information is visible inside and outside of a function.

Consider our Reynolds number calculation from before. **Question: will the following code work?**

In [3]:
```
# Defining our function
def calcReynoldsNumber(U, L, nu):
    Re = U * L / nu
    return Re

# Defining variables
U = 5
L = 10
nu = 1e-6

# Calling function
Re = calcReynoldsNumber()
print(Re)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_1060\1779576570.py in <module>
     10 
     11 # Calling function
---> 12 Re = calcReynoldsNumber()
     13 print(Re)

TypeError: calcReynoldsNumber() missing 3 required positional arguments: 'U', 'L', and
 'nu'
```

It does not because the function is expecting parameters U, L, nu.

But what about this code?

In [4]:
```python
# Defining our function
def calcReynoldsNumber():
    Re = U * L / nu
    return Re

# Defining variables
U = 5
L = 10
nu = 1e-6

# Calling function
Re = calcReynoldsNumber()
print(Re)
```

```
50000000.0
```

Scope summary

- Local scope:
  - Variables created inside function do not exist outside unless returned
  - Variables with the same name as named parameters must be passed into functions
- Global scope:
  - Variables created outside function in "main" body of script can be seen everywhere.
- Local scope trumps global scope if variables use the same name.
- Best practice: Use parameters to pass information (this makes it clear what information the function has).

A quick note: lambda functions

Something you'll often see in Python is the keyword `lambda`, followed by a line of code:

In [5]:
```python
# lambda function example

calcReynoldsNumber_alt = lambda U, L, nu: U * L / nu
calcReynoldsNumber_alt(5, 10, 1e-6)
```

Out[5]:
```
50000000.0
```

- These are called **lambda functions** or anonymous functions.
- They are the same as regular functions, but can only have one output, and can be passed as variables!