

Functions

ENDG 233 – Programming with Data

Instructor: Dean Richert



Introduction to programming
Creating and running Python programs in JupyterLab
Programming style
Variables, operators, and basic data structures
Flow control (if, else, for, etc.)



Advanced data structures
Data manipulation
Data visualization
Algorithms

Learning outcomes and outline

At the end of this lesson and accompanying active learning session you will be able to:

1. **Motivate** the use of functions in computer programming
2. **Define functions** in Python and the variants available
3. Call or **invoke functions** in Python
4. Understand some of the common **misconceptions** relative to Python functions, including the **distinction** between function definition and invocation
5. Apply your understanding of functions to decision-making using **machine learning**

Introduction

What is a function?

Ans: A chunk or block of code that **receives** information, **performs** tasks, and **reports** results back



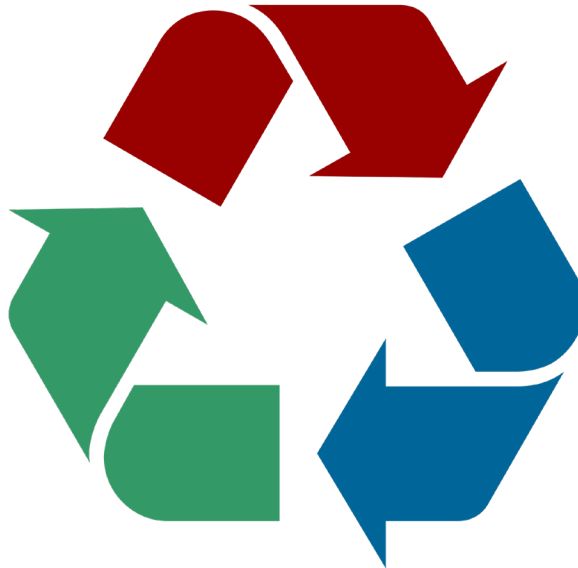
Introduction

Why do we use function in programming?

1. Code reuse

2. Modularity

3. Readability



Introduction

Why do we use function in programming?

1. Code reuse

2. Modularity

3. Readability



Introduction

Why do we use function in programming?

1. Code reuse

2. Modularity

3. Readability

```
[1]: # Create a List of numbers
numbers = [5, 10, 15, 20, 25]

# Calculate the mean of the numbers
total = 0
for num in numbers:
    total += num
mean = total / len(numbers)

# Calculate the sum of the squared differences
sum_squared_diff = 0
for num in numbers:
    diff = num - mean
    squared_diff = diff ** 2
    sum_squared_diff += squared_diff

# Calculate the variance
variance = sum_squared_diff / (len(numbers) - 1)

# Calculate the standard deviation
std_deviation = variance ** 0.5
```



```
[4]: # Create a List of numbers
numbers = [5, 10, 15, 20, 25]
```

```
[5]: # Calculate the mean of the numbers
mean = calculate_mean(numbers)
```

```
[6]: # Calculate the standard deviation
variance = calculate_variance(numbers)
```

Introduction

Analogy of a function:



Engineering example:



Function Definition

Functions must be defined **before** they can be used!

Function Definition

Functions must be defined before they can be used!

Anatomy of a function definition:

```
[ ]: def function_name(<parameters>):  
    # Perform tasks here  
    return <data>
```

Function Definition

Functions must be defined before they can be used!

Anatomy of a function definition:


```
[ ]: def function_name(<parameters>):  
    # Perform tasks here  
    return <data>
```

Function names should be lower case, meaningful, and have underscores to separate words

Function Definition

Functions must be defined before they can be used!

Anatomy of a function definition:
optional



```
[ ]: def function_name(<parameters>):  
    # Perform tasks here  
    return <data>
```

Function Definition

Functions must be defined before they can be used!


Anatomy of a function definition:

```
[ ]: def function_name(<parameters>):  
    # Perform tasks here  
    return <data>
```

Function Definition

Functions must be defined before they can be used!

Anatomy of a function definition:

```
[ ]: def function_name(<parameters>):  
     # Perform tasks here  
    return <data>
```

Function Definition

Functions must be defined before they can be used!

Anatomy of a function definition:

```
[ ]: def function_name(<parameters>):  
    # Perform tasks here  
    return <data>
```



optional

Function Definition - Example

```
[1]: def is_even(number):  
    print("determining if your number is even...")  
    if number % 2 == 0:  
        return True  
    else:  
        return False
```

Function Invocation

Functions must be called/invoked to execute its code!

```
[1]: def add_numbers(a,b):  
      sum = a + b  
      return sum
```

```
[2]: add_numbers(2,4)
```

```
[2]: 6
```

```
[3]: num1 = 3  
      num2 = 5  
      result = add_numbers(num1,num2)  
      print(result + 7)
```

```
15
```

```
[4]: char1 = 'a'  
      char2 = 'b'  
      add_numbers(char1,char2)
```

```
[4]: 'ab'
```


Function Invocation

Functions must be called/invoked to execute its code!

```
[1]: def add_numbers(a,b):  
      sum = a + b  
      return sum
```

```
[2]: add_numbers(2,4)
```

```
[2]: 6
```

```
[3]: num1 = 3  
      num2 = 5  
      result = add_numbers(num1,num2)  
      print(result + 7)
```

```
15
```

```
[4]: char1 = 'a'  
      char2 = 'b'  
      add_numbers(char1,char2)
```

```
[4]: 'ab'
```

Function Invocation

Functions must be called/invoked to execute its code!

```
[1]: def add_numbers(a,b):  
      sum = a + b  
      return sum
```

```
[2]: add_numbers(2,4)
```

```
[2]: 6
```

```
[3]: num1 = 3  
      num2 = 5  
      result = add_numbers(num1,num2)  
      print(result + 7)
```

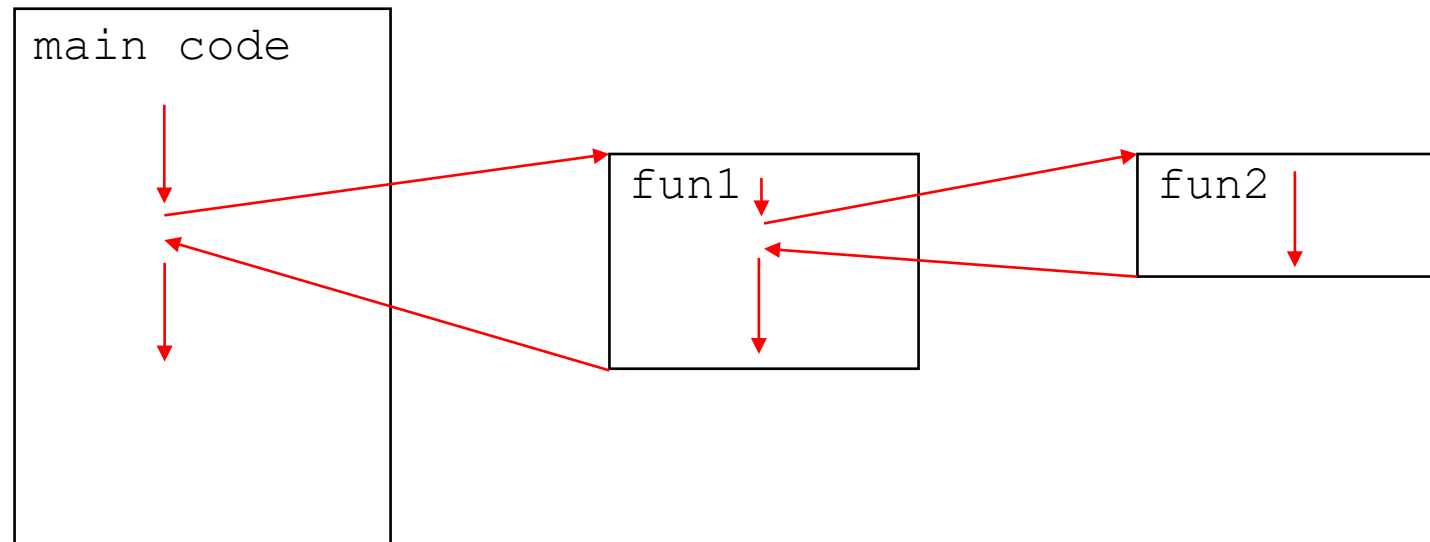
```
15
```

```
[4]: char1 = 'a'  
      char2 = 'b'  
      add_numbers(char1,char2)
```

```
[4]: 'ab'
```


Function Invocation

Functions can call other functions (or even themselves)

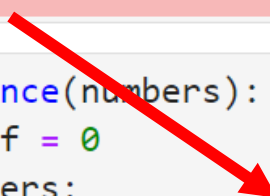


Function Invocation

Functions can call other functions (or even themselves)

$$S^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1}$$


```
[1]: def calculate_mean(numbers):  
      # Calculate the mean of the numbers  
      total = 0  
      for num in numbers:  
          total += num  
      mean = total / len(numbers)  
      return mean
```



```
[2]: def calculate_variance(numbers):  
      sum_squared_diff = 0  
      for num in numbers:  
          diff = num - calculate_mean(numbers)  
          squared_diff = diff ** 2  
          sum_squared_diff += squared_diff  
      variance = sum_squared_diff / (len(numbers) - 1)  
      return variance
```

```
[3]: calculate_variance([5, 10, 15, 20, 25])
```

```
[3]: 62.5
```

Advanced topics

Multiple return values

```
[3]: def get_statistics(data):  
      mu = calculate_mean(data)  
      var = calculate_variance(data)  
      return mu, var
```

```
[4]: mu, var = get_statistics([1,2,3,4,5])
```

```
[5]: _, var = get_statistics([1,2,3,4,5])
```

Advanced topics

Multiple return values

```
[3]: def get_statistics(data):  
      mu = calculate_mean(data)  
      var = calculate_variance(data)  
      return mu, var
```

```
[4]: mu, var = get_statistics([1,2,3,4,5])
```

```
[5]: _, var = get_statistics([1,2,3,4,5])
```

Advanced topics

Keyword arguments allow parameters to be sent out of order

```
[1]: def machine_status(pressure, temperature):  
      if pressure > 65 or temperature > 220:  
          print("warning")  
      else:  
          print("normal")
```

```
[2]: machine_status(60, 200)
```

normal

```
[3]: machine_status(temperature = 200, pressure = 60)
```

normal

Advanced topics

Default argument values allow parameters to be omitted

```
[1]: def machine_info(serial_no, mfg = "Haas"):
      if serial_no[0] == '1':
          print("This is a " + mfg + " machine from before 2010")
      else:
          print("This is a " + mfg + " machine from 2010 or after")
```

```
[2]: machine_info("123", "Fanuc")
```

This is a Fanuc machine from before 2010

```
[3]: machine_info("234")
```

This is a Haas machine from 2010 or after

Resources

- Programming with Mosh YouTube channel – [Python Functions](#) video
- W3Schools [Python Functions](#) page
 - For further study:
 - Arbitrary Arguments (`*args`)
 - Arbitrary Keyword Arguments (`**kwargs`)
 - `pass` statement
 - Recursion
- [chatGPT](#) to help with debugging or generating sample code

Knowledge Check

```
[1]: def check_the_return_values(x)  
      return x, x+2
```

```
[2]: y, _ = check_the_return_values(2)
```

- Code block #2 is an example of:
 - A) A function definition
 - B) A default argument value
 - C) A function invocation
 - D) A syntax error

Knowledge Check

```
[1]: def check_the_return_values(x)  
      return x, x+2
```

```
[2]: y, _ = check_the_return_values(2)
```

- What error is present in the code block(s) above?

Knowledge Check

```
[1]: def check_the_return_values(x):  
      return x, x+2
```

```
[2]: y, _ = check_the_return_values(2)
```

- What is the value of `y` after running code block #1 and then #2?

Active Learning Session – Getting Started



1. Download the JupyterLab Notebook from <https://bit.ly/3K6MSGr>

A screenshot of a GitHub repository page for 'functions_ALS_decisionTree'. The repository is public and has one branch and zero tags. The file list shows 'functions_ALS_decisionTrees.ipynb' and 'functions_slides.pdf'. A dropdown menu is open from the 'Code' button, showing options like 'Clone', 'Open with GitHub Desktop', and 'Download ZIP'. Red circles highlight the 'Code' button and the 'Download ZIP' option. A red arrow points from the 'Download ZIP' option to an orange box containing the text 'Extract' and 'functions_ALS_decisionTrees.ipynb'.

functions_ALS_decisionTree Public

main 1 branch 0 tags

deanmrichert Add files via upload

functions_ALS_decisionTrees.ipynb Add files via upload

functions_slides.pdf Add files via upload

Help people interested in this repository understand your project by adding

Go to file Add file <> Code

Local Codespaces New

Clone

HTTPS SSH GitHub CLI

https://github.com/deanmrichert/functions_ALS

Use Git or checkout with SVN using the web URL.

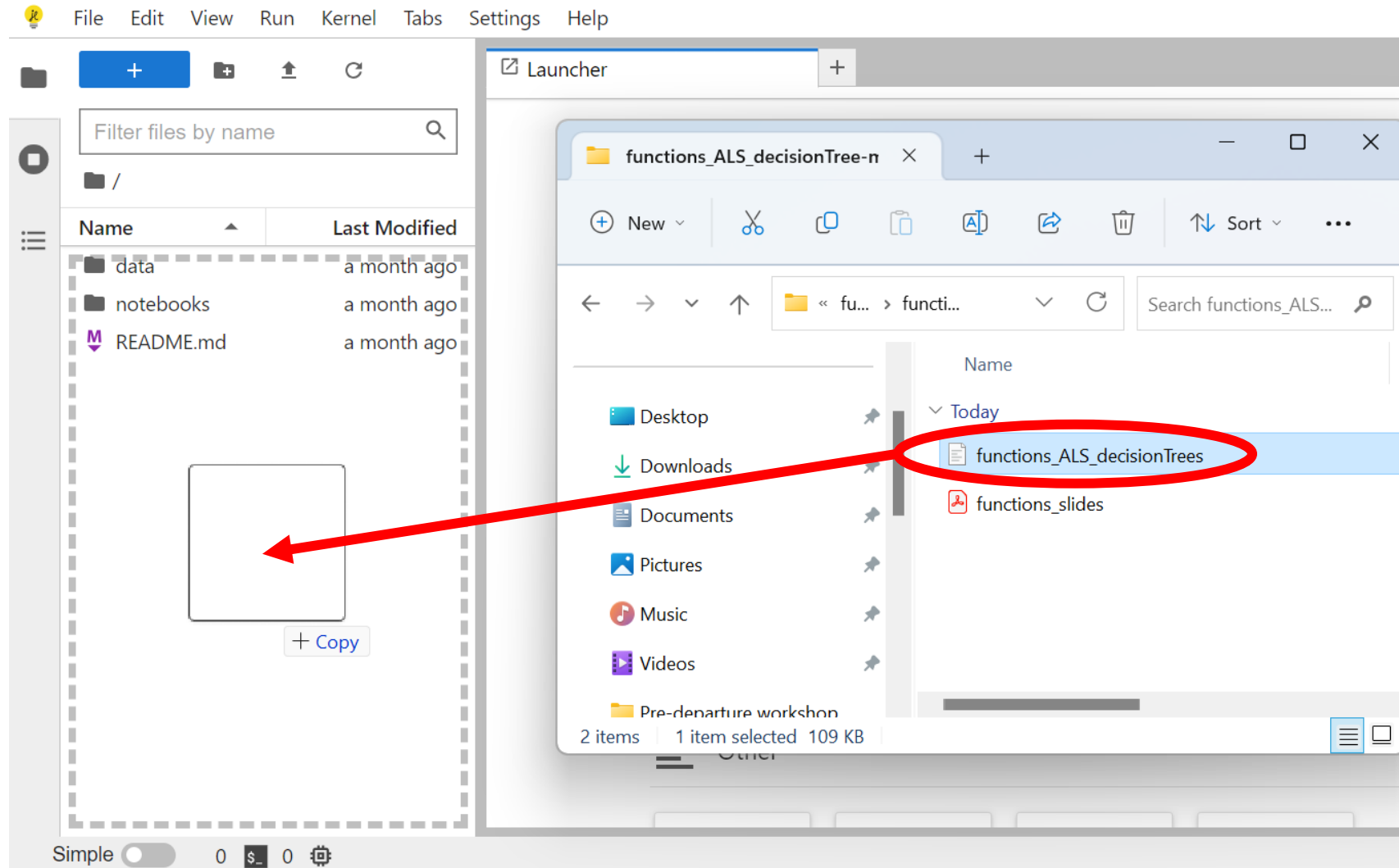
Open with GitHub Desktop

Download ZIP

Extract
functions_ALS_decisionTrees.ipynb

Active Learning Session – Getting Started

2. Open the web-based JupyterLab from <https://bit.ly/3PJrI4s>



Active Learning Session – Getting Started

3. Basic idea behind JupyterLab:

```
[3]: print("Hello World")  
Hello World  
  
[2]: a = 1  
b = 2  
a + b  
  
[2]: 3  
  
[1]: print("Did this run first?")  
Did this run first?
```

- Code cells are standalone chunks of code
- Code cells can be executed in any order, not just the order they appear
- The order in which code cells are executed are shown in square brackets [*] to the left of code cells
- Outputs appear below code cells and can be cleared by right-clicking on a cell and selecting “Clear Outputs”

Active Learning Session – Getting Started

4. General instructions for using JupyterLab

The screenshot shows the JupyterLab interface. On the left is a file browser with a search bar and a table of files. The main area displays a notebook titled 'functions_ALS_decisionTrees.ipynb'. Three red circles highlight icons in the notebook toolbar: a plus sign, a play button, and a circular arrow. Red arrows point from these icons to text boxes: 'Insert new code cell', 'Execute code cell', and 'Reset the project'.

File Browser:

Name	Last Modified
functions_A...	12 minutes ago
Intro.ipynb	a month ago
Lorenz.ipynb	a month ago
sqlite.ipynb	a month ago

Notebook Toolbar:

- Insert new code cell:** Indicated by a red circle around the '+' icon.
- Execute code cell:** Indicated by a red circle around the play button icon.
- Reset the project:** Indicated by a red circle around the circular arrow icon.

Notebook Content:

Background - Decision Trees

Decision trees are a machine learning technique used for classification or regression tasks. Classification, which is the focus of this ALS, refers to labelling an input as a member of a particular class. The goal is to label a machine as "needing maintenance" or "not needing maintenance" based on measurements from various sensors.

Decision trees have an intuitive flowchart-like structure where each node represents a feature, each branch represents a decision, and each leaf node represents an outcome or prediction. This is illustrated in the figure below which shows a decision tree that was trained to identify whether a point (x_1, x_2) is part of the red class or the blue class [1].