ENGG1811: Computing For Engineers

2025 Term 3

Functions, Lists and For-loops

Week 4: Monday 6th October, 2025

Monday 14:00 - 16:00 | HarpM15570

Today

- Functions
- Lists
- For-loops
- Lab Tips

Functions

Functions: Introduction

- We should already be familiar with functions in one form or another.
  - Examples of functions we have used in Python: `math.cos()`, `math.sin()`, `print()` .
  - Examples of functions from mathematics: $f(x) = x^2, f(x) = \ln(x), f(x, y) = \sqrt{x^2 + y^2}$.
- The goal of this lab is to get everyone comfortable creating their own functions.
  - Allows us to reuse logic without copying and pasting code.
  - Can abstract away details to make code more readable .

Functions: Introduction

- **Question:** What key details do we need to specify to define a function?
  - Inputs
  - Outputs
  - Name
  - Rule/operation
- For $f(x, y) = \sqrt{x^2 + y^2} = z$, what are the:
  - Inputs?
  - Output?
  - Name?
  - Rule/operation?

Functions: Introduction

- **Question:** What key details do we need to specify to define a function?
  - Inputs
  - Outputs
  - Name
  - Rule/operation
- For $f(x, y) = \sqrt{x^2 + y^2} = z$, what are the:
  - Inputs? x, y
  - Output? z
  - Name? Could call it distance
  - Rule/operation? Euclidean distance formula

Functions: Structure

- All functions have the same basic structure :
  - Heading : defines the function's name and inputs
  - Body : specifies the rule/operation
  - Return : specifies the output
- Sample:

```
def function(input1, input2, ...):    # Heading
    some code                          # Body
    return output1, output2, ...       # Return
```

Functions: Examples

- Suppose the only code in a file is this function:

```
def squarepluscube(num):
    square = num ** 2
    cube = num ** 3
    answer = square + cube
    return answer
```

- **Questions:**
  - What is `squarepluscube(2)` ?
  - What would be the result of `print(square)` ?

Functions: Examples

- Suppose the only code in a file is this function:

```
def squarepluscube(num):
    square = num ** 2
    cube = num ** 3
    answer = square + cube
    return answer
```

- **Questions:**
  - What is `squarepluscube(2)` ? 12
  - What would be the result of `print(square)` ?
    NameError: name 'square' is not defined

Functions: Examples

- Suppose the only code in a file is this function:

```
def multiplier(x, y, z):
    answer = x * y + z
    return answer
```

- **Questions and Answers:**
  - Let `x = 1, y = 1, z = 1` , what is:
    - `multiplier(x, y, z)` ?
    - `multiplier(x, z, y)` ?
  - Let `x = 2, y = 6, z = 3` , what is:
    - `multiplier(y, z, x)` ?
    - `multiplier(z, z, z)` ?
  - Let `v = 2, a = 0, z = 9` , what is:
    - `multiplier(v, a, z)` ?
    - `multiplier(z, v, a)` ?

Functions: Examples

- Suppose the only code in a file is this function:

```
def multiplier(x, y, z):
    answer = x * y + z
    return answer
```

- **Questions and Answers:**
  - Let `x = 1, y = 1, z = 1` , what is:
    - `multiplier(x, y, z)` ? 2
    - `multiplier(x, z, y)` ? 2
  - Let `x = 2, y = 6, z = 3` , what is:
    - `multiplier(y, z, x)` ? 21
    - `multiplier(z, z, z)` ? 12
  - Let `v = 2, a = 0, z = 9` , what is:
    - `multiplier(v, a, z)` ? 9
    - `multiplier(z, v, a)` ? 18

Functions: Examples Take Away

Take Away

The names of input variables do not matter; only the positioning matters.

- In the `multiplier` example:
  - ' `x` ' is the **first input**.
  - ' `y` ' is the **second input**.
  - ' `z` ' is the **third input**.

Remark

Placeholders used to define a function are parameters, and the actual values provided are arguments.

Lists

Lists: Introduction

- **Lists are containers for data!**
  - The container has an order, referred to as an *index*
    - First element, second element, and so on...
  - You can modify the container
    - Change a value at a specific location
    - Add or remove values
- **Questions:**
  - Is this concept useful?
  - When should we *not* use a list?

Lists: Introduction

- **Lists are containers for data!**
  - The container has an order, referred to as an *index*
    - First element, second element, and so on...
  - You can modify the container
    - Change a value at a specific location
    - Add or remove values
- **Questions:**
  - Is this concept useful? Yes, for large amounts of data
  - When should we *not* use a list? When data is unordered or mixed type

Defining Lists

- Ways to create a list:
  - Direct specification:
    - `list1 = [1, 2, 3]`
    - `list2 = [10, 100, 1000, 10000, 100000]`
  - Arithmetic operations:
    - `list3 = [6] * 3`
    - `list4 = [6] + [7] + [8]`
    - **Question:** What about '−' and '/'?
  - Appending:
    - `list5 = []`
    - `list5.append(9)`
- No restriction on list contents:
  - `list6 = ["string", 8, 9.5, [1, 2, 3], -6]`
  - **Question:** Should we mix types?

Defining Lists

- Ways to create a list:
  - Direct specification:
    - `list1 = [1, 2, 3]`
    - `list2 = [10, 100, 1000, 10000, 100000]`
  - Arithmetic operations:
    - `list3 = [6] * 3`
    - `list4 = [6] + [7] + [8]`
    - **Question:** What about '−' and '/'? Errors
  - Appending:
    - `list5 = []`
    - `list5.append(9)`
- No restriction on list contents:
  - `list6 = ["string", 8, 9.5, [1, 2, 3], -6]`
  - **Question:** Should we mix types? Not recommended; use restrictions for clarity and good practice

For-loops

For-loops: Motivations

- Loops are by *far* the most important concept in programming.
- Programming's main goal is to automate tasks we don't want to do.
  - Computers excel at performing the *same task, repeatedly*.
  - Example: squaring the first 1000 positive integers by hand—how long would it take?
    - We would likely get tired long before finishing.
    - Computers, however, never get tired or bored—they complete the task in a fraction of a millisecond.

For-loops: Introduction

- Example: squaring the first 1000 positive integers.
- Key considerations:
  - Raw value that has not yet been processed
  - Procedure to apply to this value
  - List/range of values to iterate over
- "For every integer between 1 and 1000, I want to square it and print the result."

For-loops: Structure

- All for-loops share a basic structure:
  - Heading: defines the *symbolic name* and the *list* to iterate over
  - Body: specifies the procedure to repeat
- Example:

```
for x in some_list:       # Heading
    # some code           # Body
```

- Questions:
  - Why no *return statement*?
  - Can the name `x` be changed?
  - Can I use `x` outside the loop?
  - What is the *only* thing not allowed in the body?

For-loops: Structure

- All for-loops share a basic structure:
  - Heading: defines the *symbolic name* and the *list* to iterate over
  - Body: specifies the procedure to repeat
- Example:

```
for x in some_list:       # Heading
    # some code           # Body
```

- Questions:
  - Why no *return statement*? Not a function.
  - Can the name `x` be changed? Yes.
  - Can I use `x` outside the loop? Yes, it keeps its last value.
  - What is the *only* thing not allowed in the body? Cannot permanently change `x`. Cannot redefine `some_list` (can append though, but don't do this).

For-loops: Examples

- Square integers 1 to 1000 and print:

```
for each_value in range(1, 1001):
    squared_value = each_value ** 2
    print(squared_value)
```

- Square every even integer 0 to 1000:

```
for each_value in range(0, 1001):
    if each_value % 2 == 0:
        squared_value = each_value ** 2
        print(squared_value)
```

- Alternative using step:

```
for each_value in range(0, 1001, 2):
    squared_value = each_value ** 2
    print(squared_value)
```

For-loops: Conditional Examples

- Print desk items longer than 5 letters; otherwise print “boo!”:

```
desk_items = ["keyboard", "mouse",
              "book", "glasses", "mug"]

for item in desk_items:
    if len(item) > 5:
        print(item)
    else:
        print("boo!")
```

For-loops: Using append()

- Add elements to a list with `append` :

```
list1 = [1, 2, 3, 4]
list1.append(5)
print(list1)  # [1, 2, 3, 4, 5]
```

- Combine with a loop to create new lists:

```
list2 = []
for num in list1:
    double = num * 2
    list2.append(double)
print(list2)  # [2, 4, 6, 8, 10]
```

List Comprehension

- Combine for-loops and `append` into a single line
- General structure:

```
new_list = [action(variable) for variable in old_list]
```

- Example form the previous slide:

```
list2 = [num * 2 for num in list1]
```

Lab Tips

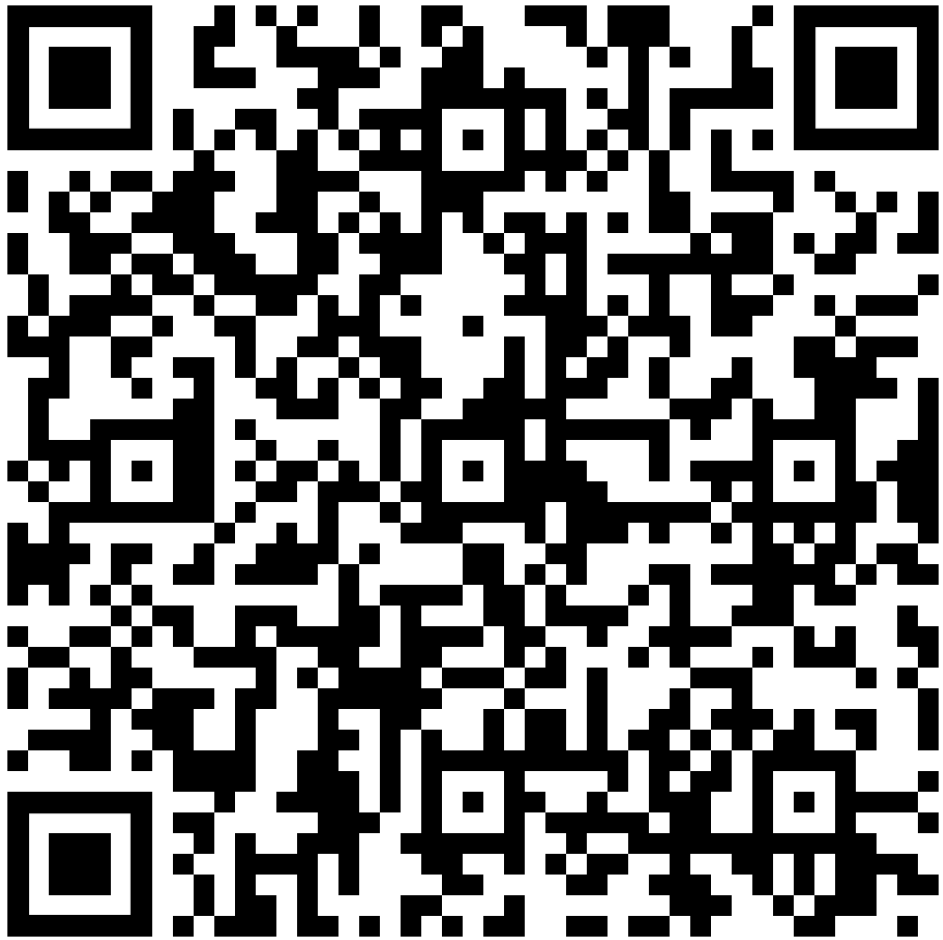
Tips

Tips for the lab today:

- Exercise 1:
  - Functions do not require input variables to have the same name.
  - You need to use the function from Task 1 inside Task 2.
- Exercise 2:
  - Use `range` and list comprehension / `list` for the horizontal list in Task 1.
  - Tasks 1 & 2 are essential for Task 3.
  - Make the vertical list in Task 3 without list comprehension or loops; any method is fine for the horizontal list.
- Exercise 3:
  - Force list is tricky: `range()` cannot use non-integer steps.
  - Read the exercise hints carefully.

Feedback

Feel free to provide anonymous feedback about the lab!



Feedback Form