# Lecture 1

## Python & Pseudocode

# Announcements

- Lab 0 on Monday
  - Refresher python exercise
- Test out attendance link
  - Regular attendance starts next week

# Warm-up Exercise

- What does this function do?

```
1 def mystery_one(numbers):
2     total = 0
3     for num in numbers:
4         if num % 2 != 0:
5             total += num
6     return total
```

# Warm-up Exercise

- What does this function do?

```
1 def mystery_two(text):
2     counts = {}
3     for char in text:
4         if char in counts:
5             counts[char] += 1
6         else:
7             counts[char] = 1
8     return counts
```

# Warm-up Exercise

- What does this function do?

```
1 def mystery_three(words):
2     return [w.upper() for w in words if len(w) > 3]
```

# Warm-up Exercise

- What does this function do?

```
1 def mystery_four(n):
2     sequence = []
3     while n != 0:
4         sequence.append(n)
5         n -= 2
6     return sequence
```

# Warm-up Exercise

- What does this function do?

```
1 def mystery_five(n):
2     steps = 0
3     while n > 1:
4         if n % 2 == 0:
5             n = n // 2
6         else:
7             n = (n * 3) + 1
8         steps += 1
9     return steps
```

# Data Types

- Atomic
  - “basic” i.e. not composed of objects of other types

# Data Types

- Atomic
  - “basic” i.e. not composed of objects of other types
  - `int, float, bool, str, NoneType`

# Data Types

- Atomic
  - “basic” i.e. not composed of objects of other types
  - `int, float, bool, str, NoneType`
- Collections
  - Composed of objects of other types

# Data Types

- Atomic
  - “basic” i.e. not composed of objects of other types
  - `int, float, bool, str, NoneType`
- Collections
  - Composed of objects of other types
  - `list, dict, set, tuple, str`

# Data Types

- Atomic
  - “basic” i.e. not composed of objects of other types
  - `int, float, bool, str, NoneType`
- Collections
  - Composed of objects of other types
  - **`list`**`, dict, set, tuple, str`
  - Lists collect objects in a given order

# Data Types

- Atomic
  - “basic” i.e. not composed of objects of other types
  - `int, float, bool, str, NoneType`
- Collections
  - Composed of objects of other types
  - `list, dict, set, tuple, str`
  - Dictionaries hold key-value pairs

# Data Types

- Atomic
  - “basic” i.e. not composed of objects of other types
  - `int, float, bool, str, NoneType`
- Collections
  - Composed of objects of other types
  - `list, dict, set, tuple, str`
  - Sets collect objects in no specific order

# Data Types

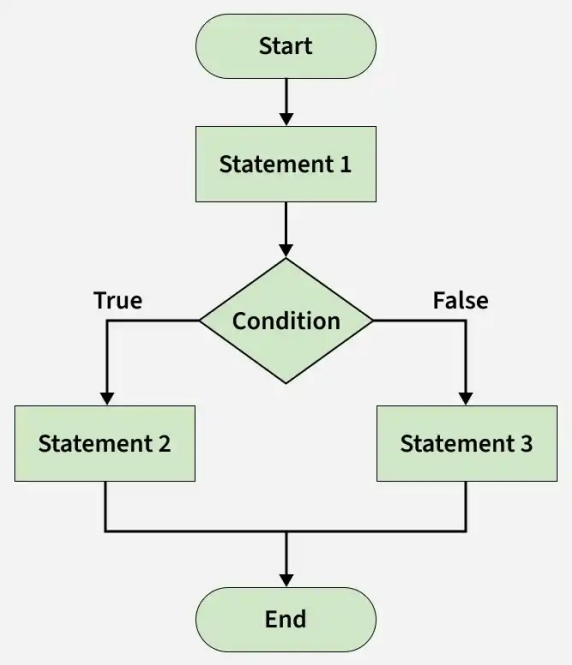
- Atomic
  - “basic” i.e. not composed of objects of other types
  - `int, float, bool, str, NoneType`
- Collections
  - Composed of objects of other types
  - `list, dict, set, tuple, str`
  - Tuples collect objects in a given order and are immutable

# Data Types

- Atomic
  - “basic” i.e. not composed of objects of other types
  - `int, float, bool, str, NoneType`
- Collections
  - Composed of objects of other types
  - `list, dict, set, tuple,` **`str`**
  - Strings collect characters in a given order and are immutable

# Control Flow

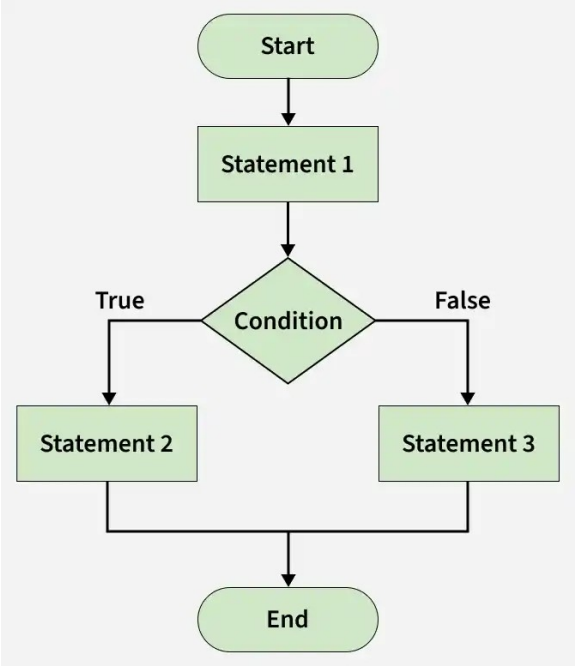
- `if-then-else`



# Control Flow

- `if-then-else`

```
1 a = 5
2 b = 1
3 if a > b:
4   print(1)
5 else:
6   print(0)
```



# Looping

- Fundamentally important to succinctly describing a process
- In Python, we have `for` loops and `while` loops

# Looping

- Fundamentally important to succinctly describing a process
- In Python, we have `for` loops and `while` loops
  - `for` loops iterate over a collection and run a code block for each object

```
1 def mystery_one(numbers):
2     total = 0
3     for num in numbers:
4         if num % 2 != 0:
5             total += num
6     return total
```

# Looping

- Fundamentally important to succinctly describing a process
- In Python, we have `for` loops and `while` loops
  - `while` loops run a code block iteratively until a condition is satisfied

```
1 def mystery_five(n):
2     steps = 0
3     while n > 1:
4         if n % 2 == 0:
5             n = n // 2
6         else:
7             n = (n * 3) + 1
8         steps += 1
9     return steps
```

# Pseudocode

- A language-agnostic, human-readable way of describing a program
- There aren’t strict rules for pseudocode
- Can look just like Python (makes Python a nice a language to use!)

# Pseudocode

- A language-agnostic, human-readable way of describing a program
- There aren’t strict rules for pseudocode
- Can look just like Python (makes Python a nice a language to use!)

```python
def mystery_one(numbers):
    total = 0
    for num in numbers:
        if num % 2 != 0:
            total += num
    return total
```

Python

```
def mystery_one(numbers):
    total = 0
    for num in numbers:
        if num is odd:
            total += num
    return total
```

Pseudocode

# Pseudocode

- A language-agnostic, human-readable way of describing a program
- There aren’t strict rules for pseudocode
- Can look just like Python (makes Python a nice a language to use!)

**Algorithm 3** Initialize $p_t$ (subroutine of Algorithm 2)

**for** $r \in \{r \in W_+ \mid |r| = 1\}$ **do**
  $t = t + 1 \quad \sigma_t \leftarrow \sigma_0 \quad r_t \leftarrow r$
  $\tilde{y}_t = M_r(D) + \mathcal{N}(0, \sigma_t^2 \mathbb{I})$
  $\rho_{used} \leftarrow \rho_{used} + \frac{1}{2\sigma_t^2}$
**end for**
$\hat{p}_t = \text{argmin}_{p \in S} \sum_{i=1}^{t} \frac{1}{\sigma_i} \left\| M_{r_i}(p) - \tilde{y}_i \right\|_2^2$

# Pseudocode

- A language-agnostic, human-readable way of describing a program
- There aren’t strict rules for pseudocode
- Can look just like Python (makes Python a nice a language to use!)

**Algorithm 2** Residuals-to-Marginals (ReM)

**Input:** Marginal workload $\mathcal{W}$, arbitrary $\mathcal{S}$, measurements $z_{\tau,i} = R_\tau p + \xi_{\tau,i}$ for $\tau \in \mathcal{S}, i = 1, \ldots, k_\tau$, where $\xi_{\tau,i}$ comes from any noise distribution

1: Estimate $\hat{\alpha}_\tau \approx R_\tau p$ for $\tau \in \mathcal{S}$ by minimizing loss function $L_\tau(\alpha_\tau)$

2: Reconstruct $\hat{\mu}_\gamma = \sum_{\tau \in \mathcal{S}: \tau \subseteq \gamma} A_{\gamma,\tau} \hat{\alpha}_\tau$ for $\gamma \in \mathcal{W}$

# Why do we need pseudocode?

- Not a trivial question!

# Why do we need pseudocode?

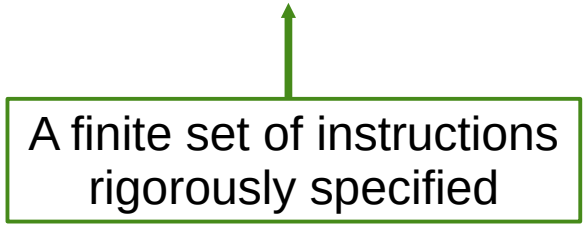
- Pseudocode abstracts from the syntax of programming languages

# Why do we need pseudocode?

- Pseudocode abstracts from the syntax of programming languages
- Pseudocode describes an **algorithm**

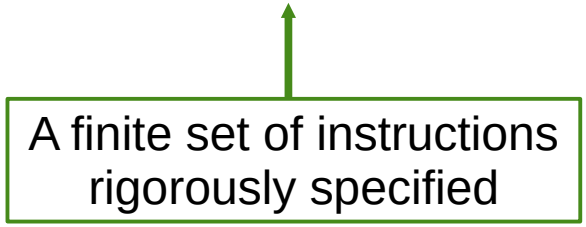
# Why do we need pseudocode?

- Pseudocode abstracts from the syntax of programming languages
- Pseudocode describes an **algorithm**



# Why do we need pseudocode?

- Pseudocode abstracts from the syntax of programming languages
- Pseudocode describes an **algorithm**




- An algorithm is an abstract mathematical object
- Pseudocode is an interface between an algorithm and its implementation

# Why do we need pseudocode?

- Pseudocode abstracts from the syntax of programming languages
- Pseudocode describes an algorithm

**Implementation** ⟷ **Pseudocode** ⟷ **Algorithm**

- An algorithm is an abstract mathematical object
- Pseudocode is an interface between an algorithm and its implementation

# Closing Questions

- Why study algorithms and not just their implementations?

# Closing Questions

- Why study algorithms and not just their implementations?

- When we say our code is efficient, are we referring to the algorithm or the implementation? Or both?