COMPSCI 590N
Lecture 2: Python Basics 2

Roy J. Adams

College of Information and Computer Sciences
University of Massachusetts Amherst

Outline

Review

- Variables and assignment

Review

- Variables and assignment
- Basic data types:

Review

- Variables and assignment
- Basic data types:
  - int

Review

- Variables and assignment
- Basic data types:
  - int
  - float

Review

- Variables and assignment
- Basic data types:
  - int
  - float
  - bool

Review

- Variables and assignment
- Basic data types:
  - int
  - float
  - bool
- Arithmetic and logical operations

Review

- Variables and assignment
- Basic data types:
  - int
  - float
  - bool
- Arithmetic and logical operations
- Sequential data types:

Review

- Variables and assignment
- Basic data types:
  - int
  - float
  - bool
- Arithmetic and logical operations
- Sequential data types:
  - list

Review

- Variables and assignment
- Basic data types:
  - int
  - float
  - bool
- Arithmetic and logical operations
- Sequential data types:
  - list
  - tuple

Review

- Variables and assignment
- Basic data types:
  - int
  - float
  - bool
- Arithmetic and logical operations
- Sequential data types:
  - list
  - tuple
  - string

Review

- Variables and assignment
- Basic data types:
  - int
  - float
  - bool
- Arithmetic and logical operations
- Sequential data types:
  - list
  - tuple
  - string
- Indexing

Review

- Variables and assignment
- Basic data types:
  - int
  - float
  - bool
- Arithmetic and logical operations
- Sequential data types:
  - list
  - tuple
  - string
- Indexing
- Dictionaries

Mutability

- A mutable object can have it's value changed. An immutable object cannot.

Mutability

- A mutable object can have it's value changed. An immutable object cannot.
- Mutable: List, Dictionary

Mutability

- A mutable object can have it's value changed. An immutable object cannot.
- Mutable: List, Dictionary
- Immutable: Int, Float, Bool, String, Tuple

Mutability

- A mutable object can have it's value changed. An immutable object cannot.
- Mutable: List, Dictionary
- Immutable: Int, Float, Bool, String, Tuple

Mutability

- A mutable object can have it's value changed. An immutable object cannot.
- Mutable: List, Dictionary
- Immutable: Int, Float, Bool, String, Tuple

DEMO

Today

- Control Flow

Today

- Control Flow
  - Conditionals

Today

- Control Flow
  - Conditionals
  - Loops

Today

- Control Flow
  - Conditionals
  - Loops
- Input and Output

Today

- Control Flow
  - Conditionals
  - Loops
- Input and Output
  - Console

Today

- Control Flow
  - Conditionals
  - Loops
- Input and Output
  - Console
  - File

Today

- Control Flow
  - Conditionals
  - Loops
- Input and Output
  - Console
  - File
- Functions

Outline

If-Elif-Else

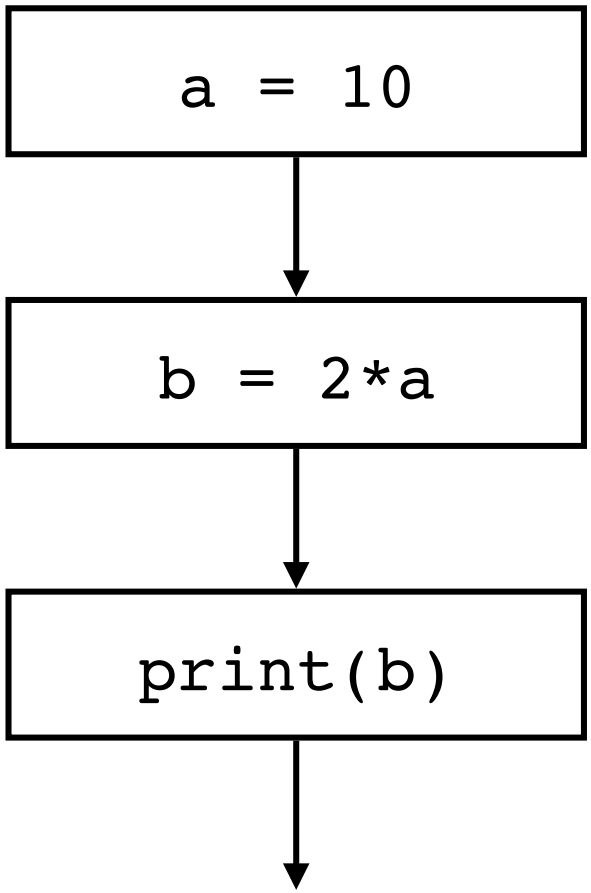
- So far: Programs as lists of instructions, executed in order.

If-Elif-Else

- So far: Programs as lists of instructions, executed in order.

If-Elif-Else

- So far: Programs as lists of instructions, executed in order.



If-Elif-Else

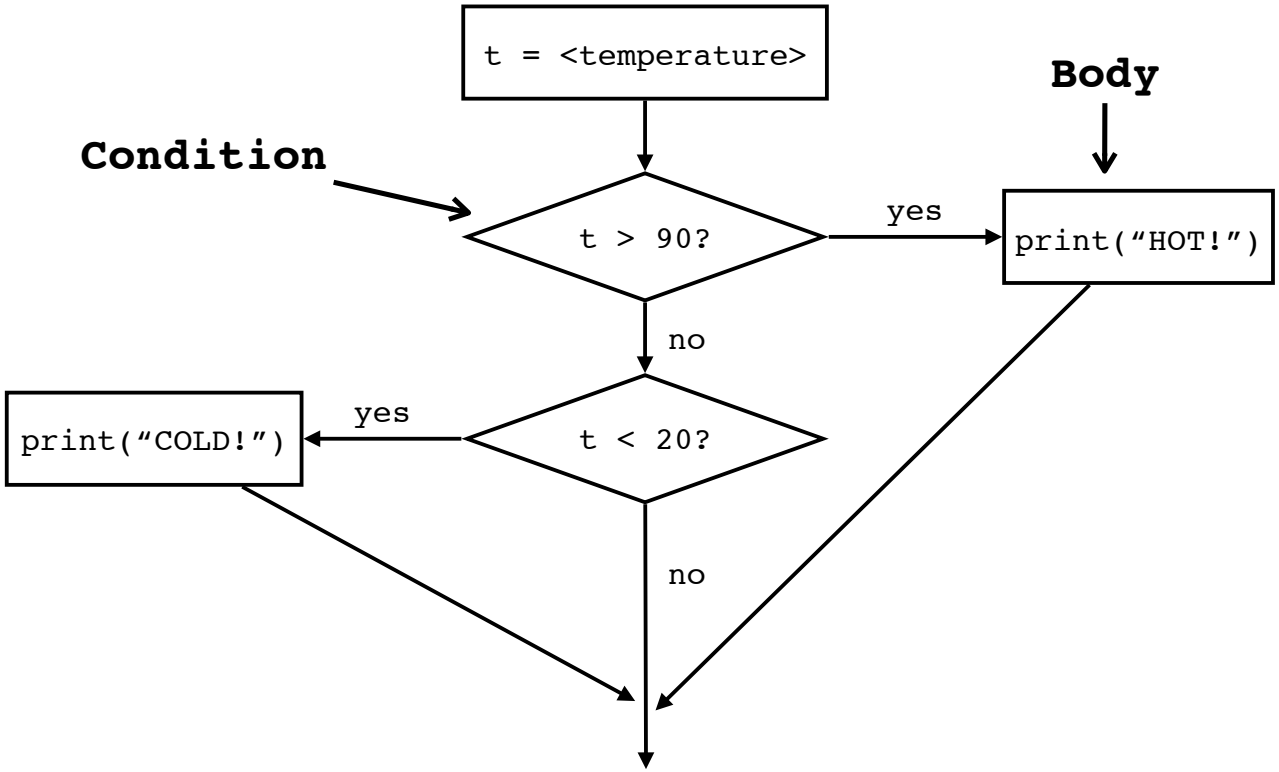
- Sequential programing is important, but cannot solve every problem.

If-Elif-Else

- Sequential programing is important, but cannot solve every problem.

If-Elif-Else

- Sequential programing is important, but cannot solve every problem.



If-Elif-Else

- If-Elif-Else statements allow us decide what code to execute depending on some conditions.

If-Elif-Else

- If-Elif-Else statements allow us decide what code to execute depending on some conditions.

If-Elif-Else

- If-Elif-Else statements allow us decide what code to execute depending on some conditions.

```
if <condition1>:
    <body1>
elif <condition2>:
    <body2>
else:
    <body3>
```

If-Elif-Else

- If-Elif-Else statements allow us decide what code to execute depending on some conditions.

```
if <condition1>:
    <body1>
elif <condition2>:
    <body2>
else:
    <body3>
```

DEMO

Loops

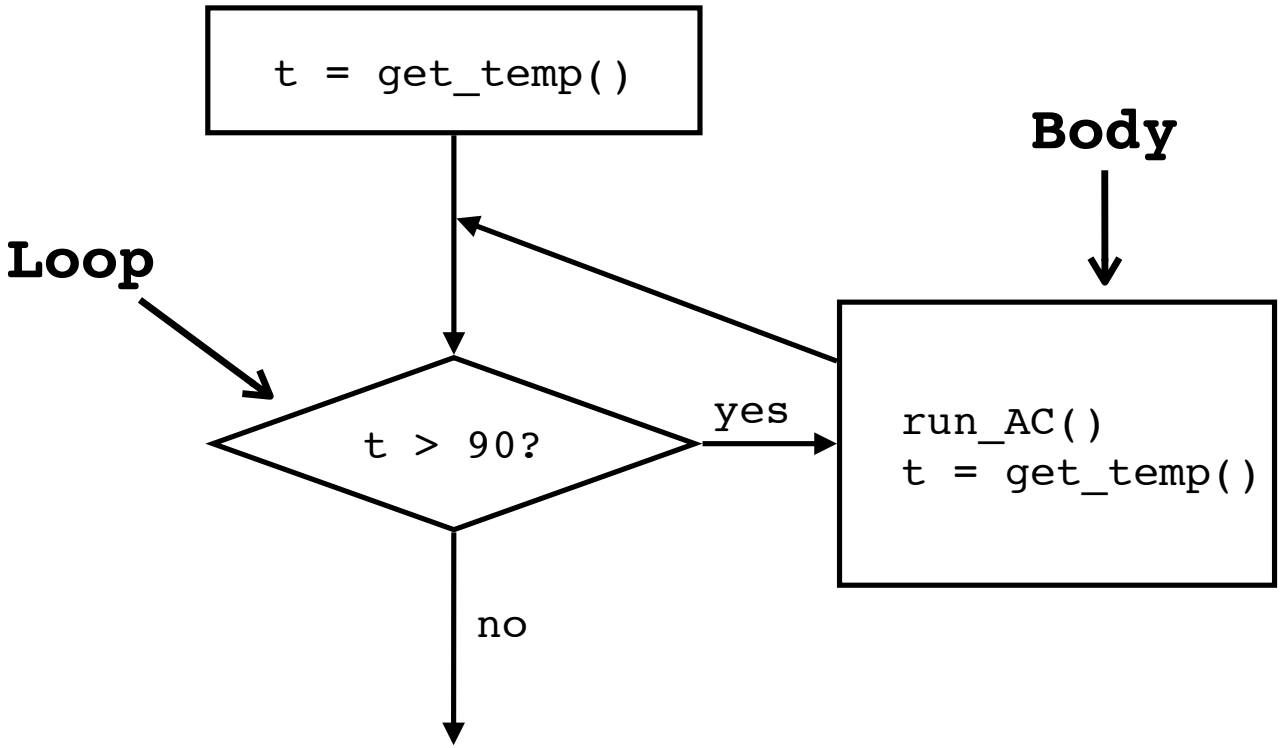
- Looping allow us run the same lines of code multiple times.

Loops

- Looping allow us run the same lines of code multiple times.

Loops

- Looping allow us run the same lines of code multiple times.



Loops

- There are two main types of loops in Python:

Loops

- There are two main types of loops in Python:
  - **For:** A block of code for every element in a sequence (list, tuple, or string).

Loops

- There are two main types of loops in Python:
  - **For:** A block of code for every element in a sequence (list, tuple, or string).

Loops

- There are two main types of loops in Python:
  - **For:** A block of code for every element in a sequence (list, tuple, or string).

```
for <var> in <sequence>:
    <body>
```

Loops

- There are two main types of loops in Python:
  - **For:** A block of code for every element in a sequence (list, tuple, or string).

```
for <var> in <sequence>:
    <body>
```

DEMO

Loops

- There are two main types of loops in Python:
  - **For:** A block of code for every element in a sequence (list, tuple, or string).
  - **While:** A block of code as long as a condition is True.

Loops

- There are two main types of loops in Python:
  - **For:** A block of code for every element in a sequence (list, tuple, or string).
  - **While:** A block of code as long as a condition is True.

```
while <condition>:
    <body>
```

Loops

- There are two main types of loops in Python:
  - **For:** A block of code for every element in a sequence (list, tuple, or string).
  - **While:** A block of code as long as a condition is True.

```
while <condition>:
    <body>
```

DEMO

Outline

What is a function?

- Functions allow you to reuse portions of code.

What is a function?

- Functions allow you to reuse portions of code.
- Functions can be thought of a sub-programs.

What is a function?

- Functions allow you to reuse portions of code.
- Functions can be thought of a sub-programs.
- Functions are *called* or *invoked*.

What is a function?

- Functions allow you to reuse portions of code.
- Functions can be thought of a sub-programs.
- Functions are *called* or *invoked*.
- Functions take *parameters* and *return a result*.

What is a function?

- Functions allow you to reuse portions of code.
- Functions can be thought of a sub-programs.
- Functions are *called* or *invoked*.
- Functions take *parameters* and *return a result*.
- We've already seen a number of built-in functions:

What is a function?

- Functions allow you to reuse portions of code.
- Functions can be thought of a sub-programs.
- Functions are *called* or *invoked*.
- Functions take *parameters* and *return a result*.
- We've already seen a number of built-in functions:
  - input

What is a function?

- Functions allow you to reuse portions of code.
- Functions can be thought of a sub-programs.
- Functions are *called* or *invoked*.
- Functions take *parameters* and *return a result*.
- We've already seen a number of built-in functions:
  - input
  - print

What is a function?

- Functions allow you to reuse portions of code.
- Functions can be thought of a sub-programs.
- Functions are *called* or *invoked*.
- Functions take *parameters* and *return a result*.
- We've already seen a number of built-in functions:
  - input
  - print
  - len

What is a function?

- Functions allow you to reuse portions of code.
- Functions can be thought of a sub-programs.
- Functions are *called* or *invoked*.
- Functions take *parameters* and *return a result*.
- We've already seen a number of built-in functions:
  - input
  - print
  - len
  - open

What is a function?

- Functions allow you to reuse portions of code.
- Functions can be thought of a sub-programs.
- Functions are *called* or *invoked*.
- Functions take *parameters* and *return a result*.
- We've already seen a number of built-in functions:
  - input
  - print
  - len
  - open
  - etc...

Defining Functions

- Functions are defined with the **def** statement.

Defining Functions

- Functions are defined with the **def** statement.

Defining Functions

- Functions are defined with the **def** statement.

```
def <function_name>(<param1>,<param2>,...):
    <body>
    return <result>
```

Defining Functions

- Functions are defined with the **def** statement.

```
def <function_name>(<param1>,<param2>,...):
    <body>
    return <result>
```

DEMO

Why use functions?

- Reuse code so you don't have to write it again.

Why use functions?

- Reuse code so you don't have to write it again.
- Provide an interface so others can use your code.

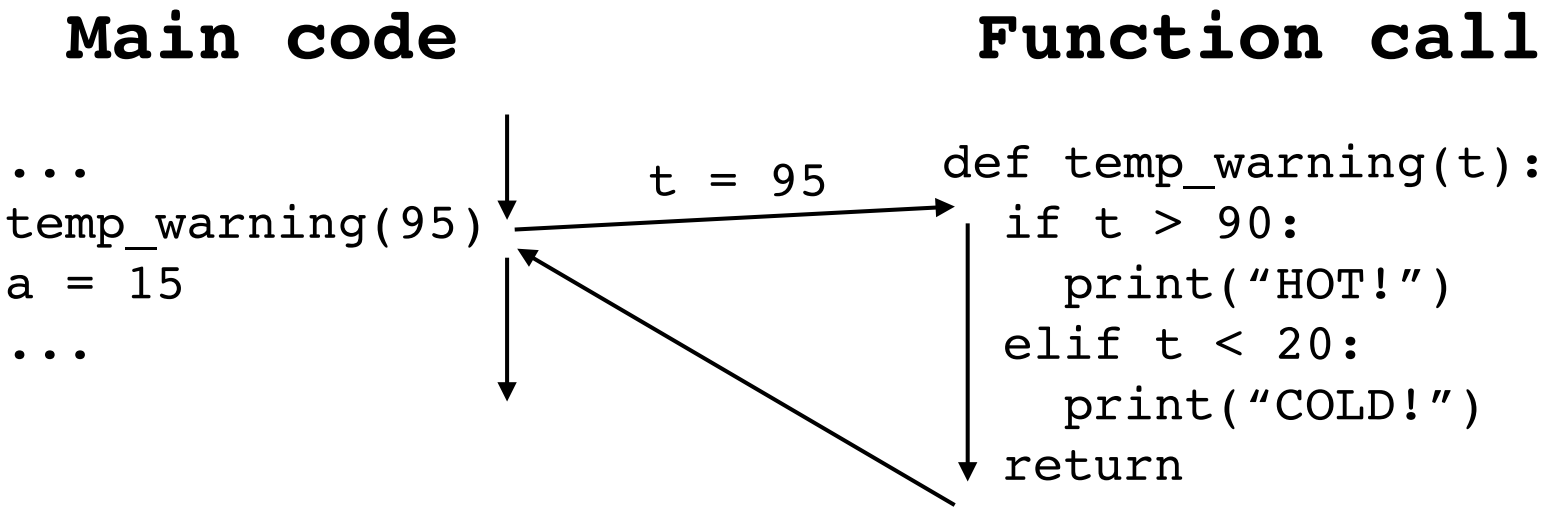
Why use functions?

- Reuse code so you don't have to write it again.
- Provide an interface so others can use your code.
- Make code more compact and readable.

Why use functions?

- Reuse code so you don't have to write it again.
- Provide an interface so others can use your code.
- Make code more compact and readable.
- **Only make changes once.**

Functions and Parameters



Outline

Interaction

- Most programs must interact either with humans or other programs.

Interaction

- Most programs must interact either with humans or other programs.
- Examples:

Interaction

- Most programs must interact either with humans or other programs.
- Examples:
  - Interacting with the console or the mouse.

Interaction

- Most programs must interact either with humans or other programs.
- Examples:
  - Interacting with the console or the mouse.
  - Reading and writing files.

Interaction

- Most programs must interact either with humans or other programs.
- Examples:
  - Interacting with the console or the mouse.
  - Reading and writing files.
  - Internet communications (we won't cover this).

Console I/O

- The easiest way to interact with the user is through the console.

Console I/O

- The easiest way to interact with the user is through the console.
- **input**: read text from the console.

Console I/O

- The easiest way to interact with the user is through the console.
- **input**: read text from the console.
- **print**: write text to the console.

Console I/O

- The easiest way to interact with the user is through the console.
- **input**: read text from the console.
- **print**: write text to the console.

Console I/O

- The easiest way to interact with the user is through the console.
- **input**: read text from the console.
- **print**: write text to the console.

```
# Prompts user and reads until <return>
user_input = input(<prompt>)

# print arguments separated by spaces
print(<print_string1>,<print_string2>,...)
```

Console I/O

- The easiest way to interact with the user is through the console.
- **input**: read text from the console.
- **print**: write text to the console.

```
# Prompts user and reads until <return>
user_input = input(<prompt>)

# print arguments separated by spaces
print(<print_string1>,<print_string2>,...)
```

DEMO

File I/O

- We can save results and interact with other programs through the file system.

File I/O

- We can save results and interact with other programs through the file system.
- **open**: open a file.

File I/O

- We can save results and interact with other programs through the file system.
- **open**: open a file.
- **read**: read the contents of a file.

File I/O

- We can save results and interact with other programs through the file system.
- **open**: open a file.
- **read**: read the contents of a file.
- **write**: write to a file.

File I/O

- We can save results and interact with other programs through the file system.
- **open**: open a file.
- **read**: read the contents of a file.
- **write**: write to a file.
- **close**: close a file.

File I/O

- We can save results and interact with other programs through the file system.
- **open**: open a file.
- **read**: read the contents of a file.
- **write**: write to a file.
- **close**: close a file.

File I/O

- We can save results and interact with other programs through the file system.
- **open**: open a file.
- **read**: read the contents of a file.
- **write**: write to a file.
- **close**: close a file.

```
file = open(<file_name>, <mode>)
file_contents = file.read()
file.write(<output>)
file.close()
```

File I/O

- We can save results and interact with other programs through the file system.
- **open**: open a file.
- **read**: read the contents of a file.
- **write**: write to a file.
- **close**: close a file.

```
file = open(<file_name>, <mode>)
file_contents = file.read()
file.write(<output>)
file.close()
```

DEMO

Importing Functions

- **Modules** are Python code, usually and collection of functions, that can be used in other programs.

Importing Functions

- **Modules** are Python code, usually and collection of functions, that can be used in other programs.
- Load modules into your code using an **import** statement.

Importing Functions

- **Modules** are Python code, usually and collection of functions, that can be used in other programs.
- Load modules into your code using an **import** statement.

Importing Functions

- **Modules** are Python code, usually and collection of functions, that can be used in other programs.
- Load modules into your code using an **import** statement.

```
import <module_name> # imports a module
```

Importing Functions

- **Modules** are Python code, usually and collection of functions, that can be used in other programs.
- Load modules into your code using an **import** statement.

```
import <module_name> # imports a module
```

DEMO