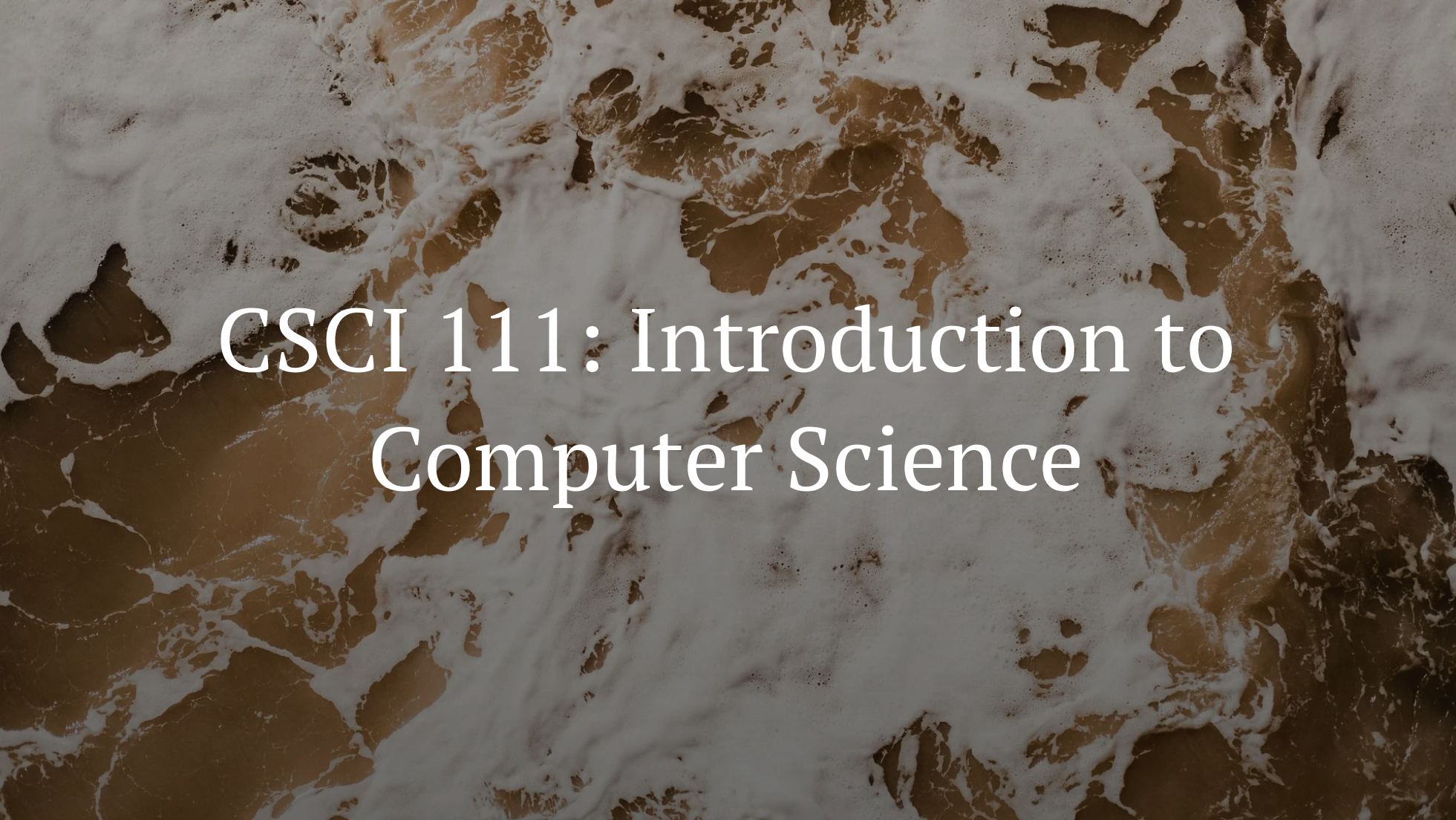
CSCI 111: Introduction to
Computer Science

Algorithm Basics

- **What is an algorithm?**
  - A step-by-step procedure for solving problems.
  - Example: Write an algorithm for making a sandwich.
- **Importance:** Algorithms are the foundation of computational problem-solving.

Creating/Implementing Algorithms

- **Key Steps:**

  1. Break down the problem.
  2. Try it out.
  3. Revise based on results.

- **Discussion:** What makes an algorithm clear for a computer to understand?
- **Exercise:** Create an algorithm to solve a simple task, such as tying your shoes.

Parts of an Algorithm

- **Input/Output:** What data you provide and what you get.
- **Operations:** The actions you can perform on the data.
- **Conditionals/Loops:** Handle special cases and repeat tasks.
- **Exercise:** Identify input, output, and conditions in your algorithm from earlier.

Why Programming Languages?

- **Motivation:** Computers don’t understand human languages like English.
- **Solution:** Programming languages provide structure and precision.
- **Python Focus:** We’ll be using Python, which is known for readability and flexibility.

What is Python?

- Python is an interpreted, high-level, general-purpose programming language.
- Created by Guido van Rossum and first released in 1991.
- Popular for its readability and flexibility.

Why Python?

- Popular and widely used.
- Focus on readability.
- A large community and plenty of libraries.
- We'll be using Python for this course.

Non-Interpreted, Low-Level Languages

Machine Code

```
10111000 00000100 00000000 00000000 00000000
10111011 00000001 00000000 00000000 00000000
10111001 00000000 10000000 00000100 00001000
10111010 00001110 00000000 00000000 00000000
11001101 10000000
10111000 00000001 00000000 00000000 00000000
10111011 00000000 00000000 00000000 00000000
11001101 10000000
```

Non-Interpreted, Low-Level Languages

Hexadecimal Representation of Machine Code

```
b8 04 00 00 00 bb 01 00 00 00 b9 00 80 04 08 ba 0e 00 00 00 cd 80
b8 01 00 00 00 bb 00 00 00 00 cd 80
```

Non-Interpreted, Low-Level Languages

x86 Assembly

```
section .data
    hello db 'Hello, World!', 0xA
    len equ $ - hello

section .text
    global _start

_start:
    mov eax, 4
    mov ebx, 1
    mov ecx, hello
    mov edx, len
    int 0x80

    mov eax, 1
    xor ebx, ebx
    int 0x80
```

Non-Interpreted, Low-Level Languages

C

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Compiled Languages

- In compiled languages like C, the source code is transformed into machine code by a compiler.
- Steps to compile a C program:
  1. **Write the source code** in a text file (e.g., `hello.c`).
  2. **Compile** the program using a compiler (e.g., `gcc`):

     ```
     gcc -o hello hello.c
     ```

  3. **Run** the compiled program:

     ```
     ./hello
     ```

- Compiled programs are generally faster than interpreted ones but require the compilation step before running.

Python as an Interpreter

- Python uses an interpreter to execute code.
- **Interactive mode**: Execute statements one by one.
- **Script mode**: Execute multiple lines as a program.
- Example:

```
>>> print("Hello, World!")
```

Syntax vs Semantics

- **Syntax**: Rules that govern the structure of valid Python programs.
- **Semantics**: What the statements mean.
- Syntax errors will stop your code from running; semantic errors may still run but produce wrong results.
- Example of a syntax error:

```
print("Hello, World!"
```

- Example of a semantic error:

```
x = 10
y = 20
average = x + y / 2
print(average)
```

Expressions and Statements

- **Expressions**: Pieces of code that produce a value.
- **Statements**: Instructions that perform an action.
- Example of an expression:

```
3 + 4
```

- Example of a statement:

```
print(3 + 4)
```

Variables and Assignment

- Variables store data.
- The assignment operator `=` assigns values to variables.
- Example:

```
x = 5
```

- **Updating variables**: Modify their values.

Precision in Python

- Python handles floating-point numbers, but they come with precision limits.
- Example:

```
print(0.1 + 0.2)
```

- Use the `round()` function to handle precision issues.

Data Types in Python

- Python supports various data types.
- **Basic types**:
  - `int` : Whole numbers.
  - `float` : Decimal numbers.
  - `str` : Text (strings).
  - `bool` : True/False.

Integers (`int`)

- Integers are whole numbers.
- Example:

```
x = 10
print(type(x))  # <class 'int'>
```

- Operations on integers:

```
a = 5
b = 3
print(a + b)
```

Floats (`float`)

- Floats are numbers with decimals.
- Example:

```
y = 3.14
print(type(y))  # <class 'float'>
```

- Example of precision issue:

```
print(0.1 + 0.2)
```

Strings (`str`)

- Strings are sequences of characters.
- Example:

```
name = "Alice"
print(type(name))  # <class 'str'>
```

- **String manipulation**:
  - Convert to uppercase: `name.upper()`
  - Concatenate strings: `first_name + last_name`

Booleans (`bool`)

- Booleans represent True or False values.
- Example:

```
is_student = True
print(type(is_student))  # <class 'bool'>
```

- **Boolean operations**:
  - Use `and`, `or`, `not` to combine conditions.

Type Conversion

- Convert between data types using `int()`, `float()`, `str()`, `bool()`.
- Example:

```
a = "10.0"
b = int(a)
```

- Example:

```
pi_str = "3.14"
pi_float = float(pi_str)
```

Input and Output

- Use the `input()` function to get user input.
- Use the `print()` function to display output.
- Example:

```
name = input("What's your name?")
print("Hello, " + name)
```

Multiple Input Examples

- Example: Getting numbers from the user

```
age = input("Enter your age: ")
age = int(age)
print("You are", age, "years old.")
```

Operators and Expressions

- **Arithmetic operators**: `+`, `-`, `*`, `/`, `**`, `%`
- **Comparison operators**: `<`, `>`, `<=`, `>=`, `==`, `!=`
- **Logical operators**: `and`, `or`, `not`
- Example:

```
x = 10
y = 5
print(x > y and y < 8)
```

Conditionals

- **If statements** allow you to execute code based on conditions.
- Example:

```
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

- Nested if statements:

```
if age > 0:
    if age >= 18:
        print("Adult")
    else:
        print("Minor")
```

Loops: For Loop

- **For loop** example:

```
for i in range(5):
    print(i)
```

- Iterate over a list:

```
colors = ["red", "green", "blue"]
for color in colors:
    print(color)
```

Loops: While Loop

- **While loop** example:

```
i = 1
while i < 5:
    print(i)
    i += 1
```

Control Statements

- **Break**: Exits the loop.
- **Continue**: Skips to the next iteration.
- Example:

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

Functions

- Functions let you package code into reusable blocks.
- Example:

```
def greet(name):
    print("Hello, " + name + "!")
```

- **Return values**:

```
def add(a, b):
    return a + b
```

More Functions: Parameters and Scope

- Functions can take parameters:

```
def square(x):
    return x * x
```

- **Scope**: Variables defined inside a function are local.
- Example of scope:

```
def my_func():
    x = 10
    print(x)

my_func()
print(x)  # Error: x is not defined
```

Review and Key Takeaways

- **Key Concepts**:
  - Syntax and Semantics.
  - Variables, Data types, Operators, Conditionals, Loops, Functions.
  - Precision handling, input/output, scope.
- **Next Steps**:
  - Practice writing Python programs.
  - Experiment with conditionals, loops, and functions.

Quiz:

Objective: Create a Python program that applies the key concepts we've learned (variables, input/output, conditionals, lists, loops, and functions) to collect and display personalized information.

Instructions:

1. **Write a Python Program** that:
   - Asks the user for their **name**.
   - Asks for their **age**.
   - Asks for a **list of their favorite hobbies**.
   - Uses a **conditional statement** to provide a custom message based on their age (e.g., "You're young!" or "You're experienced!").
   - Uses a **loop** to print each of their hobbies.
   - Stores the input in variables and **prints** a summary.

Example Interaction:

```
What is your name? Alice
How old are you? 22
What are your favorite hobbies? (separate by commas) Reading, Hiking, Gaming

Hello Alice!
You're young!
Your favorite hobbies are:
- Reading
- Hiking
- Gaming
```

Tip: Split the input string by commas to create a list of hobbies

```
hobbies = hobbies_input.split(", ")
```

Turn in on Canvas