AP Computer Science Principles

Big Idea 3A: Algorithms and Programming

Overview

In AP Computer Science Principles, **Algorithms and Programming** focuses on how programs are designed, written, tested, and improved. A program is made of instructions that tell a computer what to do. These instructions often include data, math, strings, Boolean logic, decisions, and repetition.

The main goal is to learn how to solve problems using clear, organized steps.

1. Data Abstraction

Data abstraction means using variables, lists, or other structures to store information in a way that makes a program easier to write, read, and manage.

Instead of repeating the same value many times, we can store it in a variable.

Example

```
score = 95
```

The variable score stores the value 95. Now the program can use score instead of typing 95 repeatedly.

Why Data Abstraction Matters

- Organizes information
- Reduces repeated code
- Makes programs easier to update
- Makes programs easier to understand
- Helps programs work with large amounts of data

Example Without Abstraction

```
print("Student score: 95")
print("Final grade based on 95")
```

Example With Abstraction

```
score = 95
print("Student score:", score)
print("Final grade based on", score)
```

If the score changes, we only need to update it in one place.

Lists as Data Abstraction

A **list** stores multiple values in one variable.

```
scores = [90, 85, 100, 92]
```

This list stores several student scores. Instead of making separate variables:

```
score1 = 90
score2 = 85
score3 = 100
score4 = 92
```

we can use one list.

2. Mathematical Expressions

A **mathematical expression** is code that performs a calculation.

Programs use math to solve problems, calculate totals, compare values, and update data.

Common Math Operators

Operator	Meaning	Example	Result
+	Addition	5 + 3	8
-	Subtraction	5 - 3	2
*	Multiplication	5 * 3	15
/	Division	10 / 2	5
%	Modulus / Remainder	10 % 3	1

Example

```
price = 20
tax = 2
total = price + tax
print(total)
```

Output:

```
22
```

Modulus Operator

The modulus operator % gives the **remainder** after division.

```
10 % 3
```

Result:

```
1
```

Because 10 divided by 3 is 3 remainder 1.

Common Use of Modulus

Modulus is often used to check if a number is even or odd.

```
number = 8

if number % 2 == 0:
    print("Even")
else:
    print("Odd")
```

Since 8 divided by 2 has a remainder of 0, the number is even.

3. Strings

A **string** is text data. Strings are usually placed inside quotation marks.

```
name = "Alex"
message = "Welcome to AP CSP"
```

String Concatenation

Concatenation means joining strings together.

```
firstName = "Alex"
lastName = "Garcia"

fullName = firstName + " " + lastName
print(fullName)
```

Output:

```
Alex Garcia
```

String Length

The length of a string is the number of characters in it.

```
word = "computer"
print(len(word))
```

Output:

```
8
```

Accessing Characters in a String

Many programming languages allow us to access characters using an index.

```
word = "code"
print(word[0])
```

Output:

```
c
```

Important idea: Many programming languages start counting at **0**.

Index	Character
0	c
1	o
2	d

3	e

Strings in Programs

- Names
- Passwords
- Messages
- User input
- Labels
- Text output

4. Boolean Expressions

A **Boolean expression** is an expression that evaluates to either true or false.

Boolean expressions are used when a program needs to make a decision.

Comparison Operators

Operator	Meaning	Example
==	Equal to	score == 100
!=	Not equal to	score != 0
>	Greater than	score > 70
<	Less than	score < 70
>=	Greater than or equal to	score >= 90
<=	Less than or equal to	score <= 50

Example

```
score = 85

print(score >= 70)
```

Output:

```
True
```

The expression score >= 70 is true because 85 is greater than 70.

Logical Operators

Operator	Meaning	Example
and	Both conditions must be true	age >= 16 and hasPermit == True

or	At least one condition must be true	score >= 90 or extraCredit == True
not	Reverses true/false	not isAbsent

Example Using and

```
score = 95
submitted = True

if score >= 70 and submitted == True:
    print("Passing")
```

Both conditions must be true.

Example Using or

```
hasTicket = False
hasPass = True

if hasTicket == True or hasPass == True:
    print("You may enter")
```

Only one condition needs to be true.

5. Conditionals

A **conditional** allows a program to make a decision.

The program checks a Boolean expression. If the expression is true, one section of code runs. If it is false, another section may run.

Basic If Statement

```
score = 85

if score >= 70:
    print("You passed")
```

The message prints only if the condition is true.

If / Else Statement

```
score = 65

if score >= 70:
    print("You passed")
else:
    print("You did not pass")
```

Output:

```
You did not pass
```

If / Elif / Else Statement

Used when there are more than two possible outcomes.

```
score = 85

if score >= 90:
```

```
    print("A")
elif score >= 80:
    print("B")
elif score >= 70:
    print("C")
else:
    print("Needs improvement")
```

Output:

```
B
```

Why Conditionals Matter

- Respond to user input
- Check passwords
- Determine grades
- Control game actions
- Make decisions based on data

6. Nested Conditionals

A **nested conditional** is a conditional inside another conditional.

Nested conditionals are used when a program must make a second decision after the first decision is true.

Example

```
age = 17
hasTicket = True

if age >= 16:
    if hasTicket == True:
        print("You may enter")
    else:
        print("You need a ticket")
else:
    print("You are not old enough")
```

How the Program Thinks

- Is the person at least 16?
- If yes, do they have a ticket?
- If both are true, they may enter.

Nested Conditional Example With Grades

```
score = 95
turnedIn = True

if turnedIn == True:
    if score >= 90:
        print("A")
    else:
        print("Not an A")
```

```
else:
    print("Assignment missing")
```

Nested conditionals can be useful, but too many levels can make code harder to read.

Sometimes a nested conditional can be rewritten using and.

Nested Version

```
if age >= 16:
    if hasTicket == True:
        print("Enter")
```

Combined Boolean Version

```
if age >= 16 and hasTicket == True:
    print("Enter")
```

Both versions can produce the same result.

7. Iteration / Loops

Iteration means repeating a section of code. A **loop** allows a program to run the same instructions multiple times.

Loops are useful when a task needs to be repeated.

Why Loops Matter

- Avoid repeating code
- Process lists
- Repeat actions in games
- Count numbers
- Search through data
- Keep asking for input until valid

For Loop

A **for loop** is often used when the number of repetitions is known.

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

Output:

```
Hello
Hello
Hello
Hello
Hello
```

The loop repeats 5 times.

Loop Variable

The variable i changes each time the loop repeats.

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

```
    print(i)
```

Output:

```
0
1
2
3
4
```

The loop starts at 0 and stops before 5.

While Loop

A **while loop** repeats as long as a condition is true.

```
count = 1

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

Output:

```
1
2
3
4
5
```

Infinite Loop

An **infinite loop** happens when a loop never stops.

```
count = 1

while count <= 5:
    print(count)
```

This loop never changes count, so the condition is always true.

To fix it, update the variable:

```
count = count + 1
```

Loops With Lists

Loops are often used with lists.

```
scores = [90, 85, 100]

for score in scores:
    print(score)
```

Output:

```
90
85
100
```

8. Putting the Concepts Together

Programs often use several concepts at the same time.

Example Program

```
scores = [95, 82, 67, 100]

for score in scores:
    if score >= 90:
        print("Excellent")
    elif score >= 70:
        print("Passing")
    else:
        print("Needs improvement")
```

Output:

```
Excellent
Passing
Needs improvement
Excellent
```

Concepts Used in This Program

Concept	Example
Data Abstraction	scores = [95, 82, 67, 100]
Iteration	for score in scores:
Boolean Expression	score >= 90
Conditional	if, elif, else
Mathematical / Comparison Logic	Comparing numeric values

Key Vocabulary

Term	Meaning
Algorithm	A step-by-step process for solving a problem
Program	A set of instructions that a computer follows
Variable	A named storage location for data
Data Abstraction	Using variables or lists to manage data
String	Text data
Boolean Expression	An expression that is either true or false
Conditional	Code that makes a decision
Nested Conditional	A conditional inside another conditional
Iteration	Repeating code
Loop	A programming structure that repeats instructions

Infinite Loop	A loop that never stops

Student Summary

In Big Idea 3A, students learn how programs use data and logic to solve problems. Variables and lists help store information. Mathematical expressions help programs calculate values. Strings store text. Boolean expressions allow programs to make decisions. Conditionals control which code runs. Nested conditionals handle more complex decisions. Loops allow programs to repeat actions efficiently.

These concepts are the foundation of programming in AP Computer Science Principles.