JIMMA UNIVERSITY
INSTITUTE

Bcs in Information science

Fundamental of programming 1

Name	ID
AMIRA NURI.	Ru 1244/14

Identifiers

Identifiers are names used for variabls, functions, classes, and other user-defined items in a program. Here are some examples of valid identifiers in Python:

- `my_variable`
- `myFunction`
- `my_class`
- `MyClass`
- `_my_variable`
- `my_variable_2`
- `MY_CONSTANT`

Note that identifiers cannot start with a number, and can only contain letters, numbers, and underscores. Also, it's generally a good practice to use descriptive and meaningful names for your identifiers, to make your code more readable and understandable.

Coment

A comment is a piece of text in a programthat is

ignored by the compiler or interpreter, but is used to provide information to the programmer or to document the code. In most programming languages, comments start with a special character or sequence of characters, such as "//" in Java, and "#" in Python. Here's an example of a comment in Python:

```python
# This is a comment in Python
print("Hello, World!")
```

In the above example, the first line is a comment, and it is ignored by the Python interpreter. The second line prints the text "Hello, World!" to the console.

Variables

In programming, a variable is a named location in memory that stores a value. The value can be changed during the execution of

the program. Here's an example of a variable in Python:

```python
# Declare a variable
my_variable = 42

# Print the variable
print(my_variable)

# Change the value of the variable
my_variable = "Hello, World!"

# Print the variable again
print(my_variable)
```

In the above example, we declare a variable called `my_variable` and assign it the value `42`. We then print the value of the variable to the console. Next, we change the value of the variable to the string "Hello, World!" and print it again. Variables are an essential part of programming, as they allow us to store and

manipulate data during the execution of a program.

Variable declaration

In programming, variable declaration is the act of creating a new variable and specifying its data type (if the programming language requires it). Here's an example of variable declaration in Java:

```java
// Declare a variable of type int
int myVariable;

// Declare a variable of type String and initialize it with a value
String myString = "Hello, World!";
```

In the above example, we declare a variable called `myVariable` of type `int` (which stands for integer). We do not assign it a value yet, so it will have a default value of 0.

We then declare a variable called `myString` of type `String`, which is a sequence of characters. We initialize it with the value `"Hello, World!"`.

Variable declaration is an important step in programming, as it allows us to create named locations in memory that can store values and be manipulated later in the program.

Basic data type

In programming, a data type is a classification of data that determines the type of operations that can be performed on it. Here are some examples of basic data types in Python:

```python
# Integer data type
my_integer = 42

# Floating-point data type
my_float = 3.14159
```

```
# Boolean data type
my_bool = True

# String data type
my_string = "Hello, World!"
```

In the above example, we demonstrate four basic data types: `int` (integer), `float` (floating-point number), `bool` (Boolean), and `str` (string).

`my_integer` is an example of an `int` data type, which represents whole numbers.

`my_float` is an example of a `float` data type, which represents decimal numbers.

`my_bool` is an example of a `bool` data type, which represents Boolean values (either `True` or `False`).

`my_string` is an example of a `str` data type, which represents strings of characters.

Basic data types are the building blocks of programming, and understanding them is essential for writing effective code.

Character and number

In programming, a character data type represents a single character, such as a letter, number, or symbol. A number data type represents numeric values, such as integers or floating-point numbers. Here's an example of character and number data types in C++:

```cpp
// Character data type
char my_char = 'A';
```

```
// Integer data type
int my_int = 42;

// Floating-point data type
float my_float = 3.14159;
```

In the above example, we demonstrate three data types: `char` (character), `int` (integer), and `float` (floating-point number).

`my_char` is an example of a `char` data type, and it is assigned the value `'A'`.

`my_int` is an example of an `int` data type, and it is assigned the value `42`.

`my_float` is an example of a `float` data type, and it is assigned the value `3.14159`.

It's worth noting that in some programming languages, such as Python, there is no separate character data type - a single-character string is used instead. Similarly, some languages may have multiple integer or floating-point data types with varying levels of precision.

Constant

In programming, a constant is a value that cannot be changed during the execution of a program. Once a constant is defined, its value remains the same throughout the program. Here's an example of a constant in Java:

```java
```

```
// Define a constant
final double PI = 3.14159;

// Attempt to change the value of the constant (this will result in an error)
PI = 3.14;
```

In the above example, we define a constant called `PI` and assign it the value `3.14159`. The `final` keyword is used to indicate that this variable is a constant and cannot be changed.

We then attempt to change the value of the constant to `3.14`, which is not allowed.

This will result in a compilation error, since we are attempting to modify a constant.

Constants are useful in programming when we have values that should not be changed during the execution of a program, such as mathematical constants like pi, or configuration values that should not be modified by the program itself.

Input and output

In programming, input refers to the process of receiving data from the user or from an external source, while output refers to the process of sending data to the user or to an external destination. Here's an example of input and output in Python:

```python
```

```
# Prompt the user for input
name = input("What is your name? ")

# Print a message to the console using the input value
print("Hello, " + name + "!")

# Write output to a file
with open("output.txt", "w") as file:
    file.write("Hello, " + name + "!")
```

In the above example, we prompt the user for input by using the `input()` function, which displays a message to the user and waits for them to enter a value. We store the user's input in a variable called `name`.

We then print a message to the console using the input value, by concatenating the string "Hello, " with the value of the `name` variable.

Finally, we write output to a file by using the `open()` function to create a file called "output.txt" and the `write()` function to write a message to the file.

Input and output are essential parts of programming, as they allow us to interact with the user and with external systems.

Operator

In programming, operators are symbols or keywords that perform operations on one or more operands (values or variables). Here's an example of some common

operators in JavaScript:

```javascript
// Arithmetic operators
let a = 10;
let b = 5;
let sum = a + b;        // Addition
let difference = a - b; // Subtraction
let product = a * b;    // Multiplication
let quotient = a / b;   // Division
let remainder = a % b;  // Modulo

// Comparison operators
let isEqual = a == b;   // Equality
let isNotEqual = a != b;// Inequality
```

```
let isGreaterThan = a > b;  // Greater than

let isLessThan = a < b;     // Less than

let isGreaterThanOrEqual = a >= b; // Greater than or equal to

let isLessThanOrEqual = a <= b;    // Less than or equal to


// Logical operators

let isTrue = true;

let isFalse = false;

let andResult = isTrue && isFalse; // Logical AND

let orResult = isTrue || isFalse;  // Logical OR

let notResult = !isTrue;           // Logical NOT
```

In the above example, we demonstrate three types of operators: arithmetic operators, comparison operators, and logical operators.

Arithmetic operators perform mathematical operations on operands, such as addition, subtraction, multiplication, division, and modulo (which returns the remainder of a division operation).

Comparison operators compare two values and return a Boolean value indicating whether the comparison is true or false.

Logical operators perform logical

operations on Boolean values, such as AND, OR, and NOT.

Operators are a fundamental part of programming, and understanding how to use them is essential for writing effective code.

Arithmetic operators

Arithmetic operators are symbols used in programming to perform mathematical operations on one or more operands. Here are some examples of arithmetic operators in JavaScript:

- Addition (+): adds two or more operands together. For example, `2 + 3` returns `5`.
- Subtraction (-): subtracts one operand from another. For example, `5 - 2` returns `3`.

- Multiplication (*): multiplies two or more operands together. For example, `2 * 3` returns `6`.

1. - Division (/): divides one operand by another. For example, `6 / 3` returns `2`.

- Modulus (%): returns the remainder of a division operation. For example, `7 % 3` returns `1`.

These are just a few examples of arithmetic operators in JavaScript. There are many more, depending on the programming language you are using.

Relational operators

Relational operators are symbols used in programming to compare two values and return a boolean value (either `true` or `false`) based on the comparison. Here are some examples of relational operators in Python:

- Greater than (`>`): returns `True` if the left operand is greater than the right operand. For example, `5 > 3` returns `True`.

- Less than (`<`): returns `True` if the left operand is less than the right operand. For example, `3 < 5` returns `True`.

- Greater than or equal to (`>=`): returns `True` if the left operand is greater than or equal to the right operand. For example, `5 >= 5` returns `True`.

- Less than or equal to (`<=`): returns `True` if the left operand is less than or equal to the right operand. For example, `3 <= 5` returns `True`.

- Equal to (`==`): returns `True` if the left operand is equal to the right operand. For example, `5 == 5` returns `True`.

- Not equal to (`!=`): returns `True` if the left

operand is not equal to the right operand. For example, `5 != 3` returns `True`.

These are just a few examples of relational operators in Python. There are many more, depending on the programming language you are using.

Bitwise operators

Bitwise operators are used in programming to perform operations on the binary representations of numbers. Here are some examples of bitwise operators in Java:

- Bitwise AND (`&`): performs a bitwise AND operation on two operands. For example, `5 & 3` returns `1` because the binary representation of `5` is `101` and the binary representation of `3` is `011`, and the bitwise AND of these two values is `001`.
- Bitwise OR (`|`): performs a bitwise OR operation on two operands. For example, `5 | 3` returns `7` because the binary representation of `5` is `101` and the binary representation of `3` is `011`, and the bitwise OR of these

two values is `111`.

- Bitwise XOR (`^`): performs a bitwise XOR (exclusive OR) operation on two operands. For example, `5 ^ 3` returns `6` because the binary representation of `5` is `101` and the binary representation of `3` is `011`, and the bitwise XOR of these two values is `110`.

- Bitwise complement (`~`): performs a bitwise complement operation on a single operand, which means it inverts all the bits. For example, `~5` returns `-6` because the binary representation of `5` is `00000101` (assuming a 32-bit integer), and the bitwise complement of this value is `11111010`, which represents the two's complement of `-6`.

- Left shift (`<<`): shifts the bits of the left operand to the left by the number of positions specified by the right operand. For example, `5 << 2` returns `20` because the binary representation of `5` is `101` and shifting this value to the left by two positions results in `10100`, which represents the decimal value `20`.

- Right shift (`>>`): shifts the bits of the left operand to the right by the number of positions specified by the right operand. For example, `5 >> 1` returns `2` because the binary representation of `5` is `101` and shifting this value to the right by one position results in `10`, which represents the decimal value `2`.

These are just a few examples of bitwise operators in Java. There are many more, depending on the programming language you are using.

Logical operators

Logical operators are symbols used in programming to perform logical operations on one or more boolean values. Here are some examples of logical operators in Python:

- Logical AND (`and`): returns `True` if both operands are `True`. For example, `True and False` returns `False`.
- Logical OR (`or`): returns `True` if at least one operand is `True`. For example, `True or False` returns `True`.
- Logical NOT (`not`): returns the opposite of the operand's boolean value. For example, `not True` returns `False`.

These are just a few examples of logical operators in Python. There are many more, depending on the programming language you are using. Logical operators are often used in conjunction with relational operators to

create more complex expressions. For example, `x > 5 and y < 10` returns `True` if `x` is greater than `5` and `y` is less than `10`.

Increment

. Increment operators are used in programming to increment the value of a variable by `1`. There are two types of increment operators: pre-increment and post-increment. Here are some examples of pre-increment and post-increment operators in Java:

- Pre-increment (`++i`): increments the value of the variable before it is used in an expression. For example, if `i` has the value `5`, then `++i` returns `6`.
- Post-increment (`i++`): increments the value of the variable after it is used in an expression. For example, if `i` has the value `5`, then `i++` returns `5`, but the value of `i` is now `6`.

Decrement operators

the decrement operator is used to decrease the value of a variable by 1. It is denoted by the symbol "--". Here's an example of how it works in Java:

```
int x = 5;
x--; // equivalent to x = x - 1;
System.out.println(x); // outputs 4
```

In this example, the variable `x` is initialized to 5. The `x--` statement decrements the value of `x` by 1, so `x` becomes 4. Finally, the value of `x` is printed to the console using the `System.out.println()` statement.

Similarly, you can use the decrement operator with other data types such as `float` or `double`. Here's an example in Python:

```
y = 3.5
y -= 1  # equivalent to y = y - 1
print(y)  # outputs 2.5
```

In this example, the variable `y` is initialized to 3.5. The `y -= 1` statement decrements the value of `y` by 1, so `y` becomes 2.5. Finally, the value of `y` is printed to the console using the `print()` statement.

Precedence operators

in programming, precedence refers to the order in which operators are evaluated in an expression. Operators with higher precedence are evaluated before operators with lower precedence.

Here's an example of how precedence works in

Python:

```
x = 5 + 3 * 2
print(x)  # outputs 11
```

In this example, the expression `5 + 3 * 2` is evaluated. Since the multiplication operator `*` has a higher precedence than the addition operator `+`, the expression is evaluated as `5 + (3 * 2)`, which results in `11`.

You can use parentheses to override the default precedence and force certain operations to be evaluated first. Here's an example:

```
```

```
y = (5 + 3) * 2
print(y)  # outputs 16
```

In this example, the parentheses `(5 + 3)` have the highest precedence, so the expression inside the parentheses is evaluated first. Then, the multiplication operator `*` is evaluated, which results in `16`.

It's important to be aware of operator precedence when writing complex expressions in order to ensure that the expressions are evaluated correctly. You can refer to the documentation of your programming language to learn about the specific precedence rules for the operators it supports.

Implicit type conversation

Implicit type conversation is when the type of data being used is not explicitly stated, but

rather inferred from the context of the conversation. For example, if we were talking about our favorite fruits, and I said "I love apples!", it would be understood that I am referring to the fruit type, not the technology company. Similarly, if you were to say "I need to buy more batteries", it would be inferred that you are referring to the type of battery commonly used in household devices, rather than a musical term or a legal term.

Implicit type conversation happens a lot in everyday language, and it helps us to communicate efficiently without having to constantly specify the type of data we are referring to. However, it can also lead to confusion or misunderstandings if the context is not clear, so it's important to be mindful of that as well.

Exercise from the slide

```
#include <iostream.h>

using namespace std;

int main()

{

/* Exercise 1 from Chapter 2 Write a statement (or comment) to accomplish each of the following: */

/* a) State that a program calculates the product of three integers.*/

// Calculate the product of three integers

/* b) Define the variables x, y, z, and result to be of type int.*/

 int x, y, z, result;

/* c) Prompt the user to enter three integers.*/

cout << "Please enter three integers: ";

/* d) Read three integers from the keyboard and store them in the variables x, y, and z.*/

 cin >> x >> y >> z;
```

```
/* e) Compute the product of the three integers
contained in variables x, y, and z, and assign the
result to the variable result.*/

 result = x * y * z;

/* f) Print "The product is" followed by the value of
the integer variable result. */

cout << "The product is " << result;

/* g) Return a value from main indicating that the
program terminated successfully.*/ return 0;

}

#include <iostream.h>

using namespace std;


int main() {

  int num1, num2;

  cout << "Enter two integers: ";

  cin >> num1 >> num2;

  cout << "Sum: " << num1 + num2 << endl;
```

```
  cout << "Difference: " << num1 - num2 << endl;
  cout << "Product: " << num1 * num2 << endl;
  if (num2 != 0) {
     cout << "Division: " << (double) num1 / num2 <<
endl;
  } else {
     cout << "Division: Undefined" << endl;
  }
  if (num1 > num2) {
     cout << num1 << " is greater than " << num2 <<
endl;
  } else if (num1 < num2) {
     cout << num1 << " is smaller than " << num2 <<
endl;
  } else {
     cout << num1 << " and " << num2 << " are equal"
<< endl;
  }
```

```
  return 0;
}
#include <iostream>
using namespace std;
int main()
{
double radius, circumference;

 const double PI  = 3.14159;

// define constant value of pi cout << "Enter the
radius of the circle: ";
cin >> radius;
circumference = 2 * PI * radius;

// calculate circumference
cout << "The circumference of the circle is: " <<
```

```
circumference << endl;
return 0;
}
#include <iostream>
#include <math.h>
using namespace std;

int main() {
  double a, b, c, root1, root2, discriminant;

  cout << "Enter coefficients a, b, and c: ";
  cin >> a >> b >> c;

  discriminant = b * b - 4 * a * c;  // calculate
discriminant

  if (discriminant > 0) {  // if roots are real and
```

```
different
        root1 = (-b + sqrt(discriminant)) / (2 * a);
        root2 = (-b -  sqrt(discriminant)) / (2 * a);
        cout << "The roots are real and different." <<
endl;
        cout << "Root 1 = " << root1 << endl;
        cout << "Root 2 = " << root2 << endl;
    } else if (discriminant == 0) {  // if roots are real
and equal
        root1 = -b / (2 * a);
        cout << "The roots are real and equal." << endl;
        cout << "Root 1 = Root 2 = " << root1 << endl;
    } else {  // if roots are complex
        double realPart = -b / (2 * a);
        double imagPart = sqrt(-discriminant) / (2 * a);
        cout << "The roots are complex and different."
<< endl;
        cout << "Root 1 = " << realPart << " + " <<
```

```
imagPart << "i" << endl;
      cout << "Root 2 = " << realPart << " - " <<
imagPart << "i" << endl;
  }

  return 0;
}
```

Demonstrate Chapter 3

One-way if statement

an example of a one-way if statement in Python:

```
age = 18

if age >= 18:
   print("You are old enough to vote!")
```

```

In this example, the if statement checks if the variable `age` is greater than or equal to 18. If it is, then the code inside the if statement (in this case, printing a message to the console) is executed. If `age` is less than 18, then nothing happens and the program moves on to the next line of code.

This is an example of a one-way if statement because it only executes the code inside the if statement if the condition is true. There is no code to execute if the condition is false.

## Two-way if-else statement

an example of a two-way if-else statement:

```

```
num = 10
```

```
if num % 2 == 0:
    print("The number is even.")
else:
    print("The number is odd.")
```

In this example, the program checks if the number `num` is even or odd using the modulus operator `%`. If `num` is divisible by 2 (`num % 2 == 0`), then the program prints "The number is even." Otherwise, if `num` is not divisible by 2, the program prints "The number is odd." This is a two-way if-else statement because there are two possible outcomes depending on the condition being true or false.

Multi-way if-else statement

an example of a multi-way if-else statement:

```python
# Let's say we have a variable called "age"
```

```
age = 25

# Here's a multi-way if-else statement that checks the value of "age"
if age < 18:
    print("You are not old enough to vote.")
elif age >= 18 and age <= 65:
    print("You are eligible to vote.")
else:
    print("You are eligible to vote, but may not be required to do so.")
```

In this example, the code checks the value of the variable "age" and prints a message depending on its value. If "age" is less than 18, the code will print "You are not old enough to vote." If "age" is between 18 and 65 (inclusive), the code will print "You are eligible to vote." If "age" is greater than 65, the code will print "You are eligible to vote, but may not be required to do so."

Nested if statement

an example of a nested if statement in Python:

```
x = 10
y = 5

if x > y:
    print("x is greater than y")
    if x > 15:
        print("x is also greater than 15")
    else:
        print("x is not greater than 15")
else:
    print("y is greater than x")
```

In this example, we have a nested if statement inside the first if statement. If `x` is greater than `y`, the first if statement will be true and we will enter the nested if statement. If `x` is also greater than 15, we will print "x is also greater than 15". If `x` is not greater than 15, we will print "x is not greater than 15". If `x` is not greater than `y`, we will print "y is greater than x".