COMP 1023 Introduction to Python Programming

Branching (aka Checking & Selection)

Dr. Cecia Chan, Prof. SC Cheung, Dr. Alex Lam, Dr. Desmond Tsoi

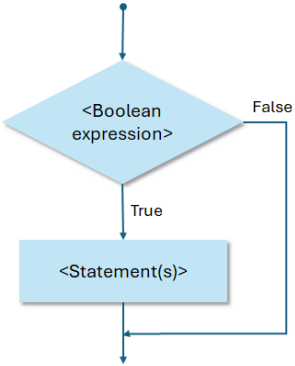
Department of Computer Science & Engineering
The Hong Kong University of Science and Technology, Hong Kong SAR, China

Why Branching (or Checking and Selection)?

- Python statements are normally executed in sequence.
- Sometimes, we need to tell the computer to do something only when certain conditions are satisfied. (Note: This alters the normal sequential execution.)
- In Python, there are three main ways to achieve this:
  - if statements
    - `if` statement
    - `if..else` statement
    - `if..elif..else` statement
    - Nested `if` statements
  - match..case statements
  - Conditional expressions (ternary operator)

The if Statement

- The `if` statement allows you to execute a statement or a block of statements when a condition is satisfied.



Syntax

```
if <Boolean expression>:
    <statement 1>  # Note that the statement must be indented
    <statement 2>  # Note that the statement must be indented
    ...
    <statement n>  # Note that the statement must be indented
```

where `<Boolean expression>` is an expression that returns a Boolean result, and `<statement 1>`, ..., `<statement n>` denote program statements that need to be executed when the `<Boolean expression>` is evaluated to True.

Example: The if Statement

```
# Filename: if_mark.py

def main():
    mark = int(input("Enter your mark: "))
    if mark >= 80:
        print("You scored A!")
    print("Your mark is", mark)

if __name__ == "__main__":
    main()
```

Output:

```
Enter your mark: 90
You scored A!
Your mark is 90
```

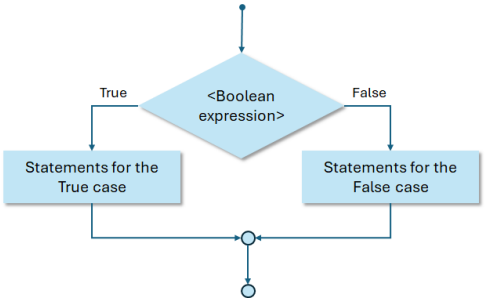
Output:

```
Enter your mark: 52
Your mark is 52
```

"You scored A!" is printed only if the input mark is 80 or higher.

The if..else Statement

- The `if..else` statement allows you to execute a statement or block of statements when a condition is satisfied or execute another statement or block of statements when the condition is not satisfied.



Syntax

```
if <Boolean expression>:
    # Note that the statements must be indented
    <statement A1>
    <statement A2>
    ...
else:
    # Note that the statements must be indented
    <statement B1>
    <statement B2>
    ...
```

where `<Boolean expression>` is an expression that returns a Boolean result, `<statement A1>`, `<statement A2>`, ... denote program statements that need to be executed when the `<Boolean expression>` is evaluated to True, and `<statement B1>`, `<statement B2>`, ... denote program statements that need to be executed when the `<Boolean expression>` is evaluated to False.

Example: The if..else Statement

```python
# Filename: if_else_temperature.py

def main():
    temperature = float(input("Body temperature: "))
    if temperature > 37.5:
        print("You are having a fever!")
    else:
        print("You have no fever.")

if __name__ == "__main__":
    main()
```

"You are having a fever!" is printed if the input temperature is higher than 37.5; otherwise, "You have no fever." is printed.

Output:

```
Body temperature: 36.9
You have no fever.
```

Output:

```
Body temperature: 37.8
You are having a fever!
```

Example: The if..else Statement

```python
# Filename: if_else_larger.py

def main():
    num1 = int(input("Enter the 1st number: "))
    num2 = int(input("Enter the 2nd number: "))
    if num1 >= num2:
        larger = num1
    else:
        larger = num2
    print("The larger number is", larger)

if __name__ == "__main__":
    main()
```

Output:

```
Enter the 1st number: 50
Enter the 2nd number: 57
The larger number is 57
```

If the first number is larger than or equal to the second number, the first number is printed; otherwise, the second number is printed.

The if..elif..else Statement (aka if-else Ladder)

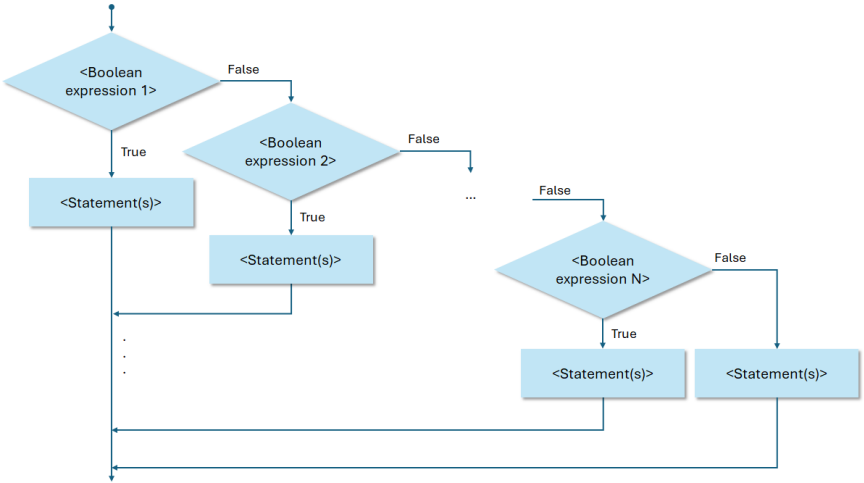
Syntax

```
if <Boolean expression 1>:
    # Note that the statement must be indented
    <statement A1>
    ...
elif <Boolean expression 2>:
    # Note that the statement must be indented
    <statement B1>
    ...
else:
    # Note that the statement must be indented
    <statement C1>
    ...
```

where <Boolean expression 1> and <Boolean expression 2> are expressions that return Boolean results. <statement A1>, ... denote program statements that need to be executed when <Boolean expression 1> is evaluated to True; <statement B1>, ... denote program statements that need to be executed when <Boolean expression 2> is evaluated to True; and <statement C1>, ... denote program statements that need to be executed when both <Boolean expression 1> and <Boolean expression 2> are evaluated to False.

Note: We may have as many elif statements in the if statement as we want.

The if..elif..else Statement (aka if-else Ladder)



Example: The if..elif..else Statement (aka if-else Ladder)

```python
# Filename: if_else_ladder_mark.py

def main():
    mark = int(input("Enter your exam mark: "))
    if mark >= 90:
        grade = 'A'
    elif mark >= 80:
        grade = 'B'
    elif mark >= 70:
        grade = 'C'
    elif mark >= 60:
        grade = 'D'
    else:
        grade = 'F'
    print("You scored", grade, "in your exam.")

if __name__ == "__main__":
    main()
```

Output:

```
Enter your exam mark: 81
You scored B in your exam.
```

Output:

```
Enter your exam mark: 59
You scored F in your exam.
```

Equivalent Statements: The if..else if..else

```
def main():  # Filename: if_else_if_else_mark.py
    mark = int(input("Enter your exam mark: "))
    if mark >= 90:
        grade = 'A'
    else:
        if mark >= 80:
            grade = 'B'
        else:
            if mark >= 70:
                grade = 'C'
            else:
                if mark >= 60:
                    grade = 'D'
                else:
                    grade = 'F'
    print("You scored", grade, "in your exam.")

if __name__ == "__main__":
    main()
```

Output:

```
Enter your exam mark: 81
You scored B in your exam.
```

Output:

```
Enter your exam mark: 59
You scored F in your exam.
```

Nested if Statement

- Both the if branch and the else branch may contain if statement(s).

Syntax

```
if <Boolean expression 1>:
    if <Boolean expression 2>:
        # Note that the statement must be indented
        <statement(s) 1>
    else:
        # Note that the statement must be indented
        <statement(s) 2>
else:
    # Note that the statement must be indented
    <statement(s) 3>
```

where <Boolean expression 1> and <Boolean expression 2> are expressions that return Boolean results; <statement(s) 1> denote program statement(s) that need to be executed when <Boolean expression 2> is evaluated to True; <statement(s) 2> denote program statement(s) that need to be executed when <Boolean expression 2> is evaluated to False; and <statement(s) 3> denote program statement(s) that need to be executed when <Boolean expression 1> is evaluated to False.

Note: The level of nested if statements can be as many as we want.

Example: Nested if Statement

```python
# Filename: nested_if_max.py

def main():
    num1 = int(input("Enter the 1st number: "))
    num2 = int(input("Enter the 2nd number: "))
    num3 = int(input("Enter the 3rd number: "))
    if num1 >= num2:
        if num1 >= num3:
            maximum = num1
        else:
            maximum = num3
    else:
        if num2 >= num3:
            maximum = num2
        else:
            maximum = num3
    print("The maximum is", maximum)

if __name__ == "__main__":
    main()
```

Output:

```
Enter the 1st number: 50
Enter the 2nd number: 99
Enter the 3rd number: 17
The maximum is 99
```

Common Errors in Selection Statements

- Consider the following code in (a) and (b): which one is correct?

```
radius = -20

if radius >= 0:
    area = radius * radius * 3.14
print("The area is", area)
```

(a) The print statement is not inside the if statement.

```
radius = -20

if radius >= 0:
    area = radius * radius * 3.14
    print("The area is", area)
```

(b) The print statement is inside the if statement.

Common Errors in Selection Statements

- Consider the following code in (a) and (b):

```
i = 1
j = 2
k = 3

if i > j:
    if i > k:
        print('A')
else:
    print('B')
```

(a) The `else` is aligned with the first `if`, so it matches the first `if`.

```
i = 1
j = 2
k = 3

if i > j:
    if i > k:
        print('A')
    else:
        print('B')
```

(b) The `else` is aligned with the second `if`, so it matches the second `if`.

- Since `i > j` is False, the code in (a) displays `B`, but nothing is displayed from the statement in (b).

match-case Statement

- The match-case statement in Python provides a way to do structural pattern matching.
- It allows for more flexible and readable conditional logic compared to traditional if-elif-else chains.
- It enables matching against data structures based on their shape and content, not just values (but we will not go into details for this).

Syntax

```
match <expression>:
    case <value 1>:
        <statement(s) 1>
    case <value 2>:
        <statement(s) 2>
    ...
    case _:
        <statement(s) N>
```

where <expression> is a value or variable to be matched against; <value 1>, <value 2>, ... are patterns that the subject is compared to; the wildcard _ refers to the default case that matches if no other case matches; and <statement(s) 1>, <statement(s) 2>, ..., <statement(s) N> are statements to be executed when the value of the <expression> matches the corresponding case.

match-case Statement

- The `<expression>` and values (i.e., `<value 1>`, `<value 2>`, ...) can be a number, a string, or a Boolean value.
- The `<value 1>`, `<value 2>`, ... must have the same data type as the value of the case expression. Note that `<value 1>`, `<value 2>` are constant expressions, meaning that they cannot contain variables in the expression, such as `1 + x`.
- When the value in a case statement matches the value of the expression, the statements starting from this case are executed.
- `case _` is the default case, which is optional. It is used to perform actions when none of the specified cases matches the expression.
- Multiple values can be combined with the pipe operator (i.e., operator `|`).
- Regular expressions (will not be covered in this course) can be used for values.

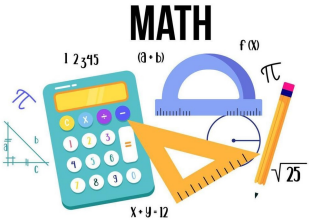
  `https://docs.python.org/3/library/re.html`
- A `None` value can be used for values.

Example: The match-case Statement

```python
def main():  # Filename: match_case_add_subtract.py
    print("Select an operation:")
    print("(A) Addition")
    print("(S) Subtraction")
    choice = input("Your choice (A or S): ")
    num1 = int(input("First integer: "))
    num2 = int(input("Second integer: "))

    match choice:
        case 'A' | 'a':  # Support multiple patterns
            result = num1 + num2
            print(f"{num1} + {num2} = {result}")
        case 'S' | 's':  # Support multiple patterns
            result = num1 - num2
            print(f"{num1} - {num2} = {result}")
        case _:
            print("Not a proper choice!")

if __name__ == "__main__":
    main()
```



Output:

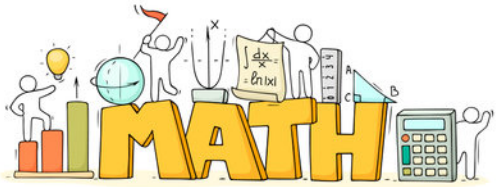
```
Select an operation:
(A) Addition
(S) Subtraction
Your choice (A or S): A
First integer: 10
Second integer: 20
10 + 20 = 30
```

Output:

```
Select an operation:
(A) Addition
(S) Subtraction
Your choice (A or S): S
First integer: 10
Second integer: 20
10 - 20 = -10
```

Output:

```
Select an operation:
(A) Addition
(S) Subtraction
Your choice (A or S): M
First integer: 10
Second integer: 20
Not a proper choice!
```



Interesting Case - Repeated Cases

- In Python, using the same value in a match-case statement does not raise an error. Instead, the cases after the first one will simply be ignored, and only the first matching case will be executed.

```python
# Filename: match_case_repeated_case.py

def main():
    value = 2
    match value:
        case 1:
            print("One")
        case 2:
            print("Two")
        case 2:  # This will be ignored
            print("Duplicate case")

if __name__ == "__main__":
    main()
```

Output:

```
Two
```

Interesting Case - Wildcard _

- In Python, placing the default case (case _) at the beginning of a match-case statement will raise an error because it makes the remaining patterns unreachable.
- The correct practice is to place the default case at the end of the match statement. This ensures that specific cases are evaluated first, allowing the default case to act as a fallback if none of the other cases match.

```
# Filename: match_case_default_beginning.py

def main():
    value = 2
    match value:
        case _: # This is a syntax error:
                # wildcard makes remaining cases unreachable
            print("Others")
        case 1: print("One")
        case 2: print("Two")

if __name__ == "__main__":
    main()
```

Comparisons of match-case vs if-elif-else

	if-elif-else	match-case
Introduced in	Available since early Python versions	Introduced in Python 3.10
Readability	Can become verbose with many conditions	More concise and readable with complex patterns
Default Case	Uses `else` for a default scenario	Uses `_` as a wildcard for the default case
Pattern Matching	Limited to simple condition checks	Supports complex pattern matching (e.g., sequences)
Performance	Generally efficient for simple conditions	Potentially more performant with complex patterns

Conditional Expressions

- Conditional expressions in Python allow us to perform conditional checks and assign values or perform operations in a single line.
- It is also known as a ternary operator.

Syntax

```
<variable name> = <expression 1> if <Boolean expression> else <expression 2>
```

where `<Boolean expression>` is a Boolean expression, `<expression 1>` and `<expression 2>` are expressions. `<variable name>` is the name of a variable to store the result. It equals either the evaluated result of `<expression 1>` or `<expression 2>`.
The `<Boolean expression>` is evaluated, and exactly one of either `<expression 1>` or `<expression 2>` is evaluated and returned based on the Boolean value of `<Boolean expression>`. If `<Boolean expression>` evaluates to True, then `<expression 1>` is evaluated and returned, while `<expression 2>` is ignored. Conversely, if `<Boolean expression>` evaluates to False, `<expression 2>` is evaluated and returned, while `<expression 1>` is ignored.

Example

```python
# Filename: conditional_expression_odd_even.py
def main():
    n = 5
    result = "even" if n % 2 == 0 else "odd"
    print("n is an", result, "number")

if __name__ == "__main__":
    main()
```

Output:

```
n is an odd number
```


```python
# Filename: if_else_odd_even.py
def main():
    n = 5
    if n % 2 == 0:
        result = "even"
    else:
        result = "odd"

    print("n is an", result, "number")

if __name__ == "__main__":
    main()
```

Output:

```
n is an odd number
```

Key Terms

- Branching statements
- Checking and selection statements
- Conditional expressions
- if statements
- if..else statements
- if..elif..else statements
- if..else ladder
- Nested if statements
- match-case statements

Review Questions

Fill in the blanks in each of the following sentences about the Python environment.

1. Selection/branching statements are used for programming with alternative courses. There are several types of selection statements: _____ statements, _____ statements, nested _____ statements, _____, and _____.
2. The various _____ statements all make control decisions based on a _____. Based on the _____ or _____ evaluation of the expression, these statements take one of the two possible courses.
3. A _____ statement matches a value with a case and executes the statements for the matched case.

Answer: 1. if, if-else, if-elif-else, match-case, conditional expressions, 2. if, Boolean expression, True, False, 3. match-case

Further Reading

- Read Sections 3.4 - 3.9, and 3.13 - 3.14 of "Introduction to Python Programming and Data Structures" textbook.

That's all!

Any questions?