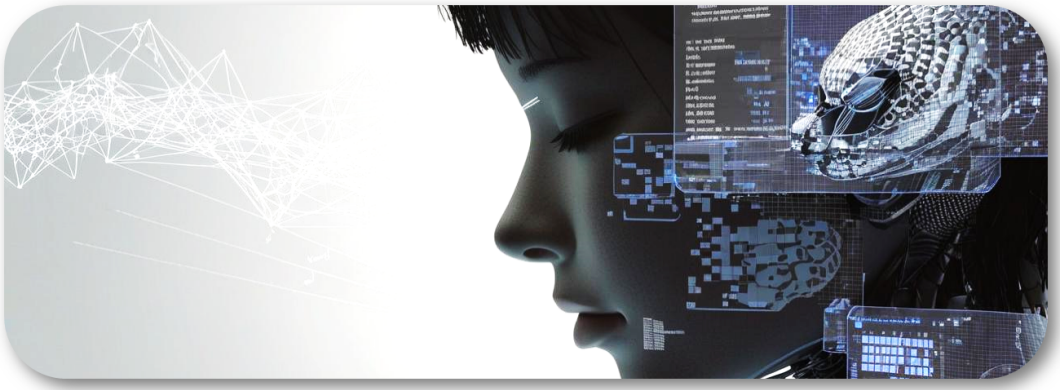

COMP 1023 Introduction to Python Programming

Tutorial 4: Branching

COMP 1023 Teaching Team

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

Why Branching (Checking and Selection)?

- Python statements are normally executed in sequence
- Branching facilitates flows of logic based on conditions
- Three main branching methods:
  - Types of `if` statements
    - `if` statements
    - `if-else` statements
    - `if-elif-else` statements
    - Nested `if`-statements
  - Match Statements
    - `match` cases
  - Conditional Expressions (Ternary Operator)

iPRS Question 1

Are these two scripts logically equivalent?

```python
# Script A
n = int(input())
if n > 50:
    if n > 75:
        print('A')
    else:
        print('B')
else:
    if n > 25:
        print('C')
    else:
        print('D')
```

```python
# Script B
n = int(input())
if n > 75:
    print('A')
elif n > 50:
    print('B')
elif n > 25:
    print('C')
else:
    print('D')
```

A. Yes

B. No

iPRS Question 1 - Answer

Are these two scripts logically equivalent?

A. Yes

B. No

Explanation

If we look at the two scripts, we will find three comparison statements: `n > 25`, `n > 50` and `n > 75`.

To deduce whether their logic are equivalent, we will need to choose 4 numbers for testing, one each from the ranges $(-\infty, 25]$, $(25, 50]$, $(50, 75]$ and $(75, \infty)$.

	$-\infty < \texttt{n} \leq 25$	$25 < \texttt{n} \leq 50$	$50 < \texttt{n} \leq 75$	$75 < \texttt{n} < \infty$
Script A	`D`	`C`	`B`	`A`
Script B	`D`	`C`	`B`	`A`

Comparison Operators

- Statement evaluates to True or False
- Common comparison operators used in evaluation:
  - `==`: Checks if two elements are equal in values
  - `!=`: Checks if two elements are not equal in values
  - `>=`, `>`, `<`, `<=`: Arithmetic comparison for numbers & Lexicographical comparison for strings
  - `in`, `not in`: Checks if an element is/is not in a sequence
  - `is`: Checks if two elements are the same object

```
a = "123"
b = "12"
print(a > b)     # True
c = "ab"
d = "abc"
print(c < d)     # True
e = "a1"
print('a' in e) # True
print('1' in e) # True
print(1 in e)    # TypeError, the left operand must be a string
```

More on Comparison Operators

- Small objects in Python are generally cached (same ID across different variable initializations). It is important to note that caching is implementation-specific.
- The `id()` function returns the ID of the variable at runtime. The ID of objects can vary in another runtime.

```
x = "a sentence for testing" # String literals are cached in a Python program
y = "a sentence for testing" # Therefore, x and y will have the same ID
print(f"id(x): {id(x)}")     # id(x): 1742992120368
print(f"id(y): {id(y)}")     # id(y): 1742992120368
a = "a" * 5000               # String multiplication results are not cached
b = "a" * 5000               # So a and b will have different IDs
print(f"id(a): {id(a)}")     # id(a): 1742994190256
print(f"id(b): {id(b)}")     # id(b): 1742994195312
print("a == b:", a == b)     # a == b: True
print("a is b:", a is b)     # a is b: False
c = a                        # c references the same object as a
print(f"id(c): {id(c)}")     # id(c): 1742994190256
print("a == c:", a == c)     # a == c: True
print("a is c:", a is c)     # a is c: True
```

Using Logic Operators with Comparison Operators

Comparison operators can be combined with and, or and not operators.
You can combine some comparison statements by chaining operators.

```
n = 5
course_code = "comp1023"
if n < 10 and n > 0 and course_code == "comp1023":
    print("Hi student! Your number is between 0 and 10")
else:
    print("Your number is either not between 0 and 10, or you're not in comp1023.")

if 0 < n < 10 and course_code == "comp1023": # Logically equivalent
    print("Hi student! Your number is between 0 and 10")
else:
    print("Your number is either not between 0 and 10, or you're not in comp1023.")
```

Using Logic Operators with Comparison Operators (Cont.)

If you have multiple conditions, you can use parentheses () to indicate precedence.

```
p, q, r = 1, 10, 55
if p >= 10 and q >= 10 or r < 100:
    print("Condition 1 fulfilled!")

if p >= 10 and (q >= 10 or r < 100):
    print("Condition 2 fulfilled!")
```

Output:

```
Condition 1 fulfilled!
```

iPRS Question 2

What Python codes should be inserted into the `if` statement such that "Nice number!" is printed only when `n` is an integer between 20 and 50 (inclusive)? Select all that apply.

```
if _____:
    print("Nice number!")
```

A. `20 <= n <= 50`

B. `n >= 20 and n <= 50`

C. `n >= 20 or n <= 50`

D. `n > 20 and n < 50`

iPRS Question 2 - Answer

What Python codes should be inserted into the `if` statement such that "Nice number!" is printed only when `n` is an integer between 20 and 50 (inclusive)? Select all that apply.

```
if _____:
    print("Nice number!")
```

A. `20 <= n <= 50`

B. `n >= 20 and n <= 50`

C. `n >= 20 or n <= 50`

D. `n > 20 and n < 50`

Explanation

- A. `20 <= n <= 50`: An integer between 20 and 50 (inclusive)
- B. `n >= 20 and n <= 50`: An integer between 20 and 50 (inclusive)
- C. `n >= 20 or n <= 50`: Incorrect, all values of `n` would evaluate to True
- D. `n > 20 and n < 50`: Incorrect, 20 and 50 are excluded.

if-else Statements

The else block is executed when the condition of the if statement evaluates to False.

```
major = input("Enter your department: ")
if major == "CSE" or major == "CPEG":
    print("You are part of CSE! Welcome to COMP 1023!")
else:
    print("You're not part of CSE, but welcome to COMP 1023 anyways!")
```

Think about this...

What happens if the condition is "`if major == "CSE" or "CPEG":`"?

if-elif-else Statements (The if-else ladder)

```
food = input("What type of food would you like for lunch? Enter here: ")
if food == "dim sum":
    print("There is a Chinese restaurant on G/F.")
elif food == "Korean":
    print("There is a Hungry Korean")
elif food == "fast food":
    print("There is a McDonald's at LG5.")
else:
    print("You can explore Canteen II and the LG7 food court!")
```

Remember that indentation must be consistent and aligned!

Nested if Statements

- Cascading `if-else` statements within another `if-else` blocks.
- These can be implemented similar to `if-elif-else` statements.

```
food = input("What type of food would you like for lunch? Enter here: ")
if food == "dim sum":
    print("There is a Chinese restaurant on G/F.")
else:
    if food == "Korean":
        print("There is a Hungry Korean")
    else:
        if food == "fast food":
            print("There is a McDonald's at LG5.")
        else:
            print("You can explore Canteen II and the LG7 food court!")
```

iPRS Question 3

Which Python statement should be inserted into the `elif` statement such that "Sweet" is printed if the two conditions below are satisfied simultaneously.

- (Condition I) `number` is an integer between 10 and 50 (inclusive)
- (Condition II) `number` is an integer divisible by 2 or divisible by 5

```
if number > 50:
    print("Too large")
elif _____________________:
    print("Sweet")
else:
    print("Okay")
```

A. `10 <= number and ( % 2 == 0 or % 5 == 0)`

B. `number >= 10 and (number % 2 == 0 and number % 5 == 0)`

C. `10 <= number or number <= 50 and (number % 2 == 0 or number % 5 == 0)`

D. `50 >= number >= 10 and (number % 2 == 0 or number % 5 == 0)`

iPRS Question 3 - Answer

Which Python statement should be inserted into the `elif` statement such that "Sweet" is printed if the two conditions below are satisfied simultaneously.

- (Condition I) `number` is an integer between 10 and 50 (inclusive)
- (Condition II) `number` is an integer divisible by 2 or divisible by 5

```
if number > 50:
    print("Too large")
elif _____________________:
    print("Sweet")
else:
    print("Okay")
```

Explanation

A. `10 <= number and ( % 2 == 0 or % 5 == 0)`: Invalid syntax

B. `number >= 10 and (number % 2 == 0 and number % 5 == 0)`: Counterexample: 15

C. `10 <= number or number <= 50 and (number % 2 == 0 or number % 5 == 0)`: Counterexample: 5

D. `50 >= number >= 10 and (number % 2 == 0 or number % 5 == 0)`

Match Statements

- Each case similar to comparing values with if statements, more flexible
- Cases are checked from top to bottom.
- It is good practice to have a final default case. It is represented by _ (an underscore).
- case matching can be more complicated statements.

```
a = 4
b = 6
match a:
    case 'a':
        print("Letter a.")
    case 1 | 2 | 3:          # 1 or 2 or 3
        print("1, 2 or 3.")
    case 4 if b % 2 == 0:    # Only checks if a matches 4 when b is an even number
        print(f"a = 4 and b is even number.")
    case x if x in "bcd":
        print("b, c or d.")
    case _:
        print("Default case reached.")
```

More on Match Statements

- Cases are checked from top to bottom.
- If there are identical cases, the first one encountered will be matched, others will be ignored.
- Match cases are type-dependent.

```
a = 1
match a:
    case "1":
        print("a is a str")
    case float(1):
        print("a is a float")
    case 1:
        print("a is an int")
    case _:
        print("Default case")
```

Output:

```
a in an int
```

Conditional Expressions (Ternary Operators)

```
preference = "juicy"

fruit1 = "apple" if preference != "juicy" else "orange"
if preference != "juicy":
    fruit2 = "apple"
else:
    fruit2 = "orange"

print(fruit1 == fruit2) # True
```

Syntax of Conditional Expressions

```
<expression1> if <condition> else <expression2>
```

Chained Conditional Expressions (Nested Ternary Operators)

You can turn `if-elif-else` statements into a one-line statement by using nested conditional expressions. However, you are not recommended to do so, as it greatly reduces readability of the code.

```
preference = "red"

fruit1 = "orange" if preference == "orange" else ("banana" if preference == "yellow" else
↪  ("pear" if preference == "green" else "apple"))

if preference == "orange":
    fruit2 = "orange"
elif preference == "yellow":
    fruit2 = "banana"
elif preference == "green":
    fruit2 = "pear"
else:
    fruit2 = "apple"
print(fruit1 == fruit2) # True
```

Using Conditional Expressions in `print` Statements

Here is an example for fun (Writing code this way is not recommended):

```
a = "red"
print("apple" if a == "red" else "orange")
```

Output:

```
apple
```

iPRS Question 4

What Python code should be inserted into the `s =` statement such that "Hello" is printed when `n` is equal to 5 and "Okay" is printed otherwise.

```
n = 5
s = _____________________
print(s)
```

A. `"Hello" else "Okay"`

B. `"Hello" if n == 5 else "Okay"`

C. `if n == 5: "Hello"; else: "Okay"`

iPRS Question 4 - Answer

What Python code should be inserted into the `s =` statement such that "Hello" is printed when `n` is equal to 5 and "Okay" is printed otherwise.

```
n = 5
s = _____________________
print(s)
```

Explanation

A. `"Hello" else "Okay"`: Invalid syntax

B. `"Hello" if n == 5 else "Okay"`

C. `if n == 5: "Hello"; else: "Okay"`: Invalid syntax

Practice Question 1: Nested `if` Statements

Question: What is the output of the following program?

```
a, b = 1, 2
if a > 0:
    if b > 3:
        print("X")
else:
    print("Y")
```

Practice Question 1: Answer

Answer

Nothing is printed.

- `a > 0` is `True`, so we enter the outer `if` block
- `b > 3` is `False`, so we skip printing "X"
- The `else` corresponds to the outer `if` statement which was evaluated to `True` (`a > 0`), so the `else` block is skipped

Practice Question 2: Complex `elif` Chain

Question: What is the output of the following program?

```
score, bonus, extra = 74, True, 5
if score >= 90:
    grade = "A+"
elif score >= 80:
    grade = "A" if bonus else "A-"
elif score >= 70:
    adjusted = score + (extra if bonus and score < 80 else 0)
    grade = "A-" if adjusted >= 80 else "B+"
else:
    grade = "F"

result = f"Excellent! {grade}" if "A" in grade else f"Good work! {grade}" \
    if "B" in grade else f"Keep trying! {grade}"
print(result)
```

Practice Question 2: Answer

Answer

Output: `Good work! B+`

- `score < 90 and score < 80`, so third `elif` executes
- `adjusted = 79 = 74 + 5` (add `extra` because `bonus = True` and `score < 80`)
- `adjusted >= 80` is `False`, so `grade = "B+"`
- in the `result` string assignment, `"A" in grade` evaluated to `False`, so the `else` part is executed, `"B" in grade` is evaluated to `True`. Therefore, `result` is `"Good work! B+"`.

Practice Question 3: Chained Conditional Expressions

Question: What is the output when `x = 3`?

```
result = "A" if x > 5 else "B" if x > 0 else "C"
print(result)
```

Practice Question 3: Answer

Answer

Output: `B`

- `x > 5` is `False`, so not "A"
- `x > 0` is `True`, so result is "B"
- Ternary operators are evaluated from left to right
- Equivalent to:

```
if x > 5:
    result = "A"
elif x > 0:
    result = "B"
else:
    result = "C"
```

Practice Question 4: Match-Cases with Guards

Note: Tuples are to be covered in later lectures

Question: What is the output of this program?

```
data = (3, 4)
match data:
    case (x, y) if x > y:
        print(f"First: {x}")
    case (x, y) if x + y == 7:
        print(f"Sum: {x + y}")
    case (x, y) if x < y:
        print(f"Third: {y}")
    case _:
        print("No match")
```

Practice Question 4: Answer

Answer

Output: `Sum: 7`

- First case: `3 > 4` is `False`
- Second case: `3 + 4 == 7` is `True`
- Match-case stops at first matched pattern
- The third case is never reached

Practice Question 5: Short Circuit Evaluation

Question: What happens when `a = 0, b = 5`?

```
a = 0
b = 5
result = "X" if a and b/a > 2 else "Y" if b else "Z"
print(result)
```

Practice Question 5: Answer

Answer

Output: `Y`

- `a and b/a > 2`: Because `a = 0`, `a` is evaluated as `False` in the condition, short-circuit evaluation skips the other part of `and` operator (`b/a`)
- No `ZeroDivisionError` because `b/a` is never evaluated
- In the second condition check: `b` = 5 which is considered `True` because not equal to zero, so `result` assigned `"Y"`
- Key concept: `and` operator stops in the first part evaluated to `False` (from left to right)

Practice Question 6: Chained Comparisons

Question: What prints when `x = 5`?

```
x = 5
if False == x == 5:
    print("A")
elif False == (x == 5):
    print("B")
elif (False == x) == 5:
    print("C")
else:
    print("D")
```

Practice Question 6: Answer

Answer

Output: `D`

- `False == x == 5` → `(False == x) and (x == 5)` → `(False == 5) and (5 == 5)` → `False and True` → `False`
- Key concept: While it makes sense to "expand" the chained condition, `x` is only evaluated once
- `False == (x == 5)` → `False == True` → `False`
- `(False == x) == 5` → `False == 5` → `False`
- All `if` conditions are evaluated to `False`, so `D` is printed

That's all!

Any questions?