Lecture 2: Decision Making in Python

INF 605 · Fall 2026 · 2026-09-01

Open in Colab → **File → Save a copy in Drive**. You cannot edit the class original.

Last class: `print`, arithmetic, variables, types, and f-strings. Today a program chooses a path with `True` / `False`, `if`, `elif`, `else`, and `and` / `or` / `not`.

Start-of-class quiz

Type answers in the Canvas attendance quiz (empty boxes only). The questions stay here.

1. Last class, what is the difference between `/` and `//` on numbers?
2. Write one `print` that uses an f-string and a variable from last class (name, a number, or a string).

Boolean values

Every `if` asks a yes/no question. The answer is `True` or `False` (capital T and F). Last class listed `bool` as a type; today that type runs the rest of the notebook.

In [4]:
```
is_student = True
has_graduated = False
print("is_student:", is_student)
print("has_graduated:", has_graduated)
print(type(True))
```

```
is_student: True
has_graduated: False
<class 'bool'>
```

`type(True)` is `<class 'bool'>`. There is no third value such as "maybe".

Comparison operators

A comparison is an expression. Python evaluates it and you get `True` or `False`.

In [7]:
```
age = 20
print("age >= 18:", age >= 18)
print("age < 18:", age < 18)
```

```
age >= 18: True
age < 18: False
```

`age >= 18` with `age = 20` is `True`. Store that result in a variable if you want to print it later.

Numbers:

In [10]:
```
a = 10
b = 15
print("a == b:", a == b)
print("a != b:", a != b)
print("a < b:", a < b)
print("a > b:", a > b)
print("a <= b:", a <= b)
print("a >= b:", a >= b)
```

```
a == b: False
a != b: True
a < b: True
a > b: False
a <= b: True
a >= b: False
```

`==` tests equality. `=` assigns. `a == b` is `False` here because 10 is not 15.

Strings compare character by character:

In [13]:
```
print("Alice" == "Alice")
print("Alice" == "alice")
print("apple" < "banana")
```

```
True
False
True
```

`"Alice" == "alice"` is `False`. Case matters.

Exercise (comparisons)

For `n = 7`, print `True` or `False` for each of these (one print per line):

1. `n` is greater than 5
2. `n` is even (`n % 2 == 0`)
3. `n` is not equal to 10

In [ ]:
```
n = 7
# print the three True/False results
```

Solution

In [18]:
```
n = 7
print(n > 5)
```

```
print(n % 2 == 0)
print(n != 10)
```

```
True
False
True
```

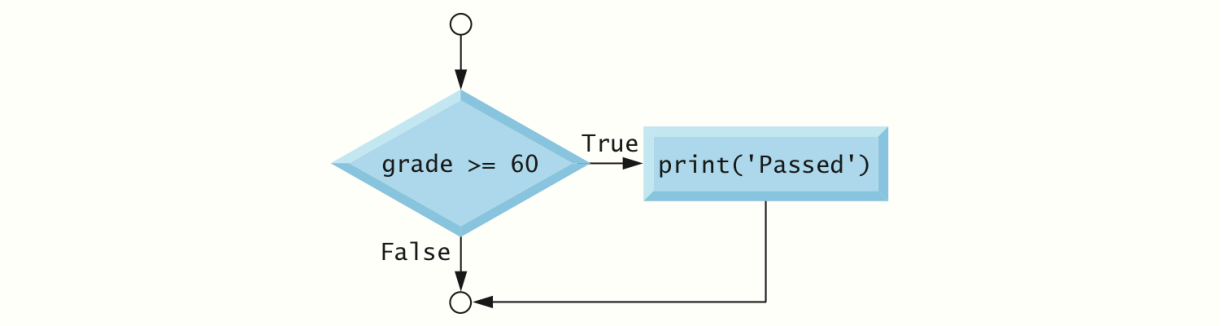
The `if` statement

If the condition is `True`, Python runs the indented block. If it is `False`, Python skips that block. The colon and the indent are required.

Textbook flowchart (Deitel): the True path prints `Passed`; the False path does nothing extra.

In [20]:
```
# FIGURE: if_statement.png
print("if flowchart: True runs print('Passed'); False skips it.")
```

```
if flowchart: True runs print('Passed'); False skips it.
```



Textbook example: a passing grade is 60.

In [22]:
```
grade = 85
if grade >= 60:
    print("Passed")
```

```
Passed
```

`85 >= 60` is `True`, so `Passed` prints.

In [24]:
```
age = 20
if age >= 18:
    print("You are an adult.")
    print("You can vote.")
print("This line always runs.")
```

```
You are an adult.
You can vote.
This line always runs.
```

Both indented lines run. The last line is not indented, so it always runs.

In [26]:
```
age = 15
if age >= 18:
```

```
    print("You are an adult.")
print("This line always runs.")
```

```
This line always runs.
```

`15 >= 18` is `False`, so the indented print is skipped. The unindented line still runs.

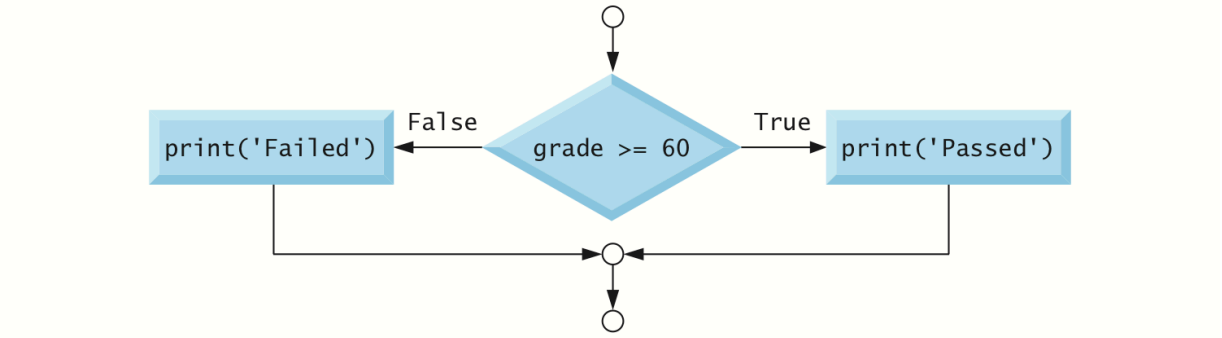
The `if` … `else` statement

One path if the condition is `True`, the other if it is `False`. Exactly one of the two blocks runs.

Textbook flowchart (Deitel):

In [29]:
```
# FIGURE: if_else.png
print("if-else flowchart: True prints Passed; False prints Failed.")
```

```
if-else flowchart: True prints Passed; False prints Failed.
```



Same passing grade of 60, now with an `else` path.

In [31]:
```
grade = 85
if grade >= 60:
    print("Passed")
else:
    print("Failed")
```

```
Passed
```

85 is at least 60, so `Passed`.

In [33]:
```
grade = 57
if grade >= 60:
    print("Passed")
else:
    print("Failed")
```

```
Failed
```

57 is below 60, so `Failed`.

In [35]:
```
number = 23
if number % 2 == 0:
```

```
    print(number, "is even")
else:
    print(number, "is odd")
```

```
23 is odd
```

`23 % 2` is 1, so this number is odd.

The `elif` chain

More than two outcomes: check from the top. The first `True` condition wins; the rest are skipped. Put the tighter test first (`>= 90` before `>= 80`).

In [38]:
```
score = 87
if score >= 90:
    letter_grade = "A"
elif score >= 80:
    letter_grade = "B"
elif score >= 70:
    letter_grade = "C"
elif score >= 60:
    letter_grade = "D"
else:
    letter_grade = "F"
print(score, letter_grade)
```

```
87 B
```

87 is not `>= 90`, then it is `>= 80`, so `B`. Python does not keep checking after that.

In [40]:
```
age = 25
if age < 13:
    category = "Child"
elif age < 18:
    category = "Teenager"
elif age < 65:
    category = "Adult"
else:
    category = "Senior"
print(f"Age {age} is classified as: {category}")
```

```
Age 25 is classified as: Adult
```

25 is not under 13, not under 18, and is under 65, so `Adult`.

Exercise (elif)

`grade` is a number 0–100. Set `letter` and `message`:

- 90–100: A, Excellent!
- 80–89: B, Good work!
- 70–79: C, Satisfactory

- 60–69: D, Needs improvement
- below 60: F, Please see instructor

Print `grade`, `letter`, and `message`.

In [ ]:
```
grade = 87
letter = None
message = None
# if-elif-else on grade
print("Grade:", grade)
print("Letter:", letter)
print("Message:", message)
```

Solution

In [45]:
```
grade = 87
if grade >= 90:
    letter = "A"
    message = "Excellent!"
elif grade >= 80:
    letter = "B"
    message = "Good work!"
elif grade >= 70:
    letter = "C"
    message = "Satisfactory"
elif grade >= 60:
    letter = "D"
    message = "Needs improvement"
else:
    letter = "F"
    message = "Please see instructor"
print("Grade:", grade)
print("Letter:", letter)
print("Message:", message)
```

```
Grade: 87
Letter: B
Message: Good work!
```

`and`, `or`, `not`

`and`: both sides True. `or`: at least one True. `not`: flips True/False.

In [47]:
```
age = 20
has_id = True
print("can buy alcohol:", age >= 21 and has_id)
print("can enter building:", age >= 18 and has_id)
```

```
can buy alcohol: False
can enter building: True
```

20 is not `>= 21`, so the alcohol line is `False` even with an ID.

In [49]:
```
is_student = True
is_senior = False
age = 20
print("discount:", is_student or is_senior or age < 12)
```

```
discount: True
```

`is_student` is True, so the whole `or` chain is True.

In [51]:
```
is_raining = False
is_weekend = True
print("good day for a walk:", (not is_raining) and is_weekend)
```

```
good day for a walk: True
```

`not False` is `True`, and it is the weekend, so the walk line is True.

Exercise (and)

Roller coaster: at least 48 inches tall **and** at least 8 years old. Print whether this person can ride.

In [ ]:
```
height = 52  # inches
age = 10
can_ride = None  # height >= 48 and age >= 8
print("Height:", height)
print("Age:", age)
print("Can ride:", can_ride)
```

Solution

In [56]:
```
height = 52
age = 10
can_ride = height >= 48 and age >= 8
print("Height:", height)
print("Age:", age)
print("Can ride:", can_ride)
if can_ride:
    print("Enjoy the ride.")
else:
    print("Sorry, you cannot ride.")
```

```
Height: 52
Age: 10
Can ride: True
Enjoy the ride.
```

`input()` (simulated here)

`input()` pauses and returns a **string**, even if the person typed digits. Convert with `int()` or `float()` before doing math.

Colab would wait for a person to type. This notebook uses the same values as strings so every demo can run without typing.

In [58]:
```
age_text = "25"          # same as: age_text = input("Enter your age: ")
age = int(age_text)
print("next birthday:", age + 1)
```

```
next birthday: 26
```

`"25"` becomes the integer 25, then adding 1 is legal.

In [60]:
```
price_text = "19.99"     # same as: price_text = input("Enter the price: ")
price = float(price_text)
print("with 8 percent tax:", round(price * 1.08, 2))
```

```
with 8 percent tax: 21.59
```

Use `float` when the number can have a decimal.

Nested `if` (short map)

An `if` inside another `if`. Read the outer question first, then the inner one. Lecture 3 is loops; do not spend the rest of today nesting.

In [63]:
```
age = 20
has_license = True
if age >= 18:
    if has_license:
        print("You can drive.")
    else:
        print("Old enough, but no license yet.")
else:
    print("Too young to drive.")
```

```
You can drive.
```

Outer test: 20 is at least 18. Inner test: there is a license, so this person can drive.

Comprehensive exercises

Comprehensive exercise 1

Movie ticket:

- under 12: child, $8
- 12–64: adult, $12
- 65 and up: senior, $10

Print the category and the price for `age = 35`.

In [ ]:
```
age = 35
category = None
price = None
# if-elif-else on age
print("Customer age:", age)
print("Category:", category)
print("Ticket price:", price)
```

Solution

In [68]:
```
age = 35
if age < 12:
    category = "Child"
    price = 8
elif age < 65:
    category = "Adult"
    price = 12
else:
    category = "Senior"
    price = 10
print("Customer age:", age)
print("Category:", category)
print("Ticket price:", price)
```

```
Customer age: 35
Category: Adult
Ticket price: 12
```

Comprehensive exercise 2

A cafe: tea is $3.50. A student with `n_teas` cups pays with a $50 bill. If the total is at most 50, print the change. Otherwise print that they need more money.

In [ ]:
```
n_teas = 7
tea_price = 3.50
cash = 50
total = None
# total = n_teas * tea_price
# then if total <= cash: print change, else: print that they need more
print("total:", total)
```

Solution

In [72]:
```
n_teas = 7
tea_price = 3.50
cash = 50
total = n_teas * tea_price
print("total:", total)
if total <= cash:
    print("change:", cash - total)
else:
    print("need more money")
```

```
total: 24.5
change: 25.5
```

Takeaways

1. `True` and `False` are the only boolean values. Comparisons such as `>=` produce them.
2. `if` runs a block only when the condition is True. `else` is the other path. `elif` is extra tests, top to bottom.
3. Indentation is the block. A line with less indent is not inside the `if`.
4. `and` needs both sides True. `or` needs one. `not` flips the value.
5. `input()` returns a string. Convert with `int` or `float` before arithmetic.
6. Next class: `for`, `while`, and `range`.