Controlling Program Flow

The selection structure

- A selection structure evaluates a condition, which is an expression that’s true or false
- This allows you to specify different courses of action based on the evaluation
  - Do one thing if true
  - Do something else if not (false)

The Selection Structure

- A branch is a sequence of statements that are only executed if a specific condition is met
  - This branch may never get executed in your program

Boolean expression

- A selection structure depends on a **condition**
  - This is an expression describing the relationship between **two** values that's evaluated *when* it appears in program code
- A Boolean expression is one that evaluate to `true` or `false`.
  - Named after George Boole who developed an extensive system of logic based on *true* and *false* conditions and their consequences

Booleans and Equality Operator

- Booleans are used to compare values
  - Are you old enough to drive?
  - Is the correct username entered?
  - Did I successfully retrieve the information from the server?

To see if an answer is equivalent to an expected value we use the equality operator = =

Using this returns a value of either *true* or *false*

NOTE: Many students confuse the mathematical operator = to the equality operator ==

Remember that = is the assignment operator in coding.

Relational Operators

- The == is a common relational operator. Here are some additional relational operators that are common in programming

Symbol	Meaning
<	Less than
>	Greater than
==	Equal to
<=	Less than or equal to
>=	Greater than or equal to
!=	Not equal to

Relational operators

• `age < 60`	Checks to see if the age is less than 60
• `hours > 40`	Checks to see if the hours are greater than 40
• `region == "Ontario"`	Checks if the region is equal to Ontario
• `status != "denied"`	Checks if the status is NOT denied
• `quantity <= 10`	Checks if the quantity is 10 and under (includes 10)
• `grade >= 90`	Checks if the grade is greater than or equal to 90

- NOTE : The relational operator for checking if two values are equal is a double ==. A single equals sign is the assignment operator.

Controlling Program Flow

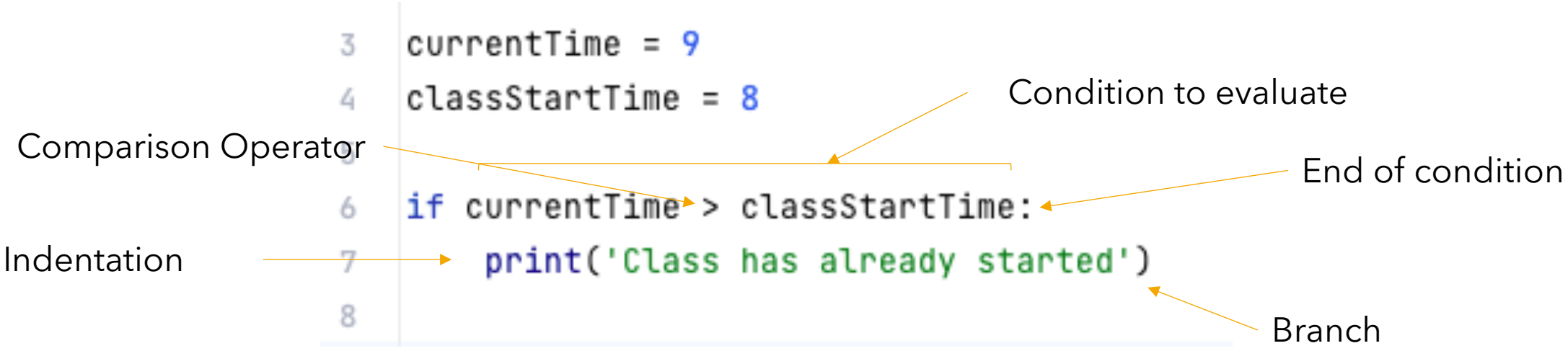
- The most common way of controlling a programs flow is to determine what a program will do based on decisions.
- These decisions can be simple of complex
- The decision process is facilitated using an if statement

If statement

- The simplest selection structure is one in which an action is take ***IF*** a condition evaluates to true, but no action will be taken if the condition evaluates to false
- This is a ***single-outcome section***

If statement

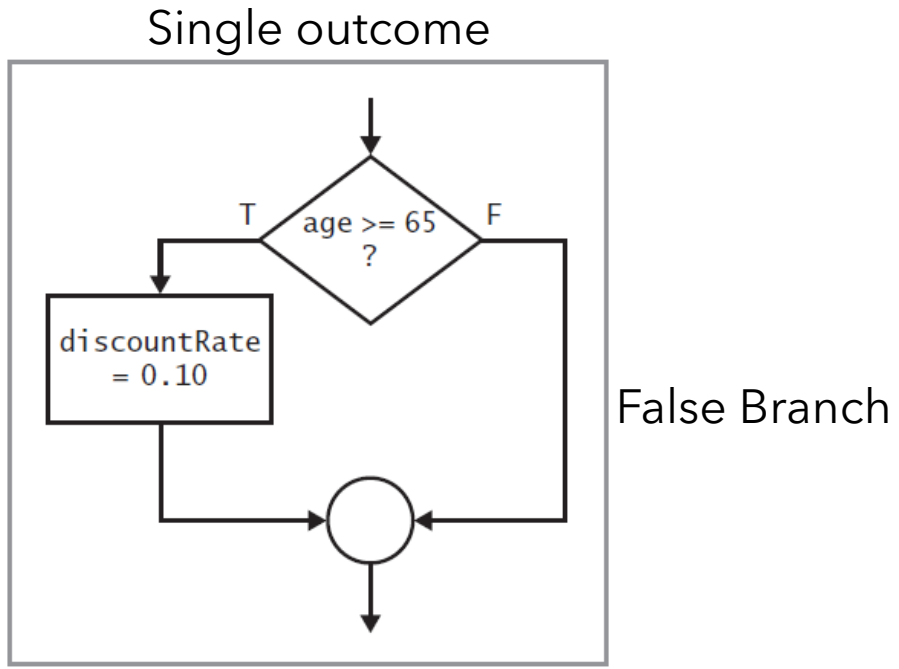
- Python syntax



NOTE: An indentation is a tab – it is not a series of spaces.

Flow Charting

- The diamond shape is the standard symbol for flow charting.



Following evaluation, the program branches merge and the program continues

Dual Outcome

- A dual outcome is a selection statement where you will perform one set of instructions if a condition evaluates to true , otherwise you will perform a second set of instructions
- Dual outcomes work under the principles of Boolean logic – something is either true or false – if true do this, otherwise it has to be false so do the other step
- Dual outcomes work with the keywords if and else

If this is true ,do this, ***else*** do this

Dual Outcome

```
2
3  currentTime = 8
4  classStartTime = 9
5
6  if currentTime > classStartTime:
7      print('Class has already started!')
8  else:
9      print('You can still make it to class!')
10
```

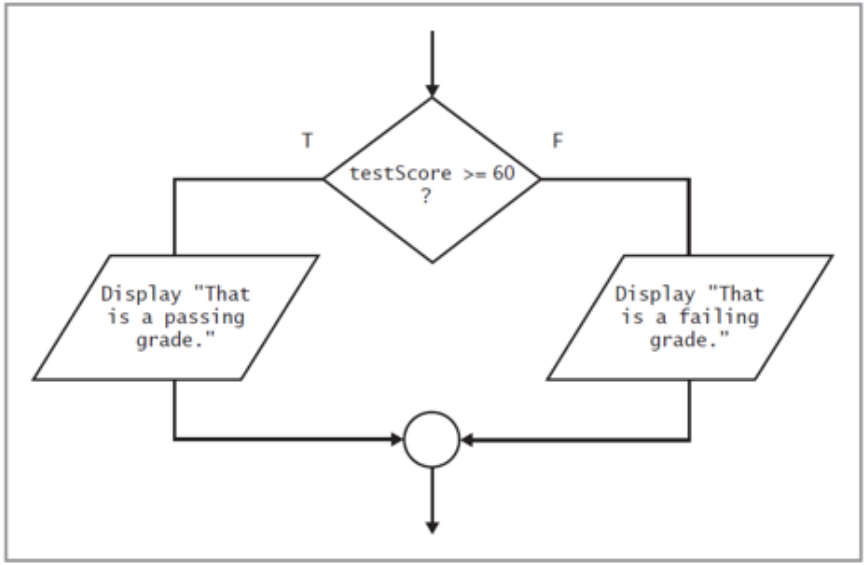
Indentation

This only runs if the condition evaluates to true

This only runs if the condition evaluates to false

Flowcharting

True Branch



False Branch

Multiple Outcomes

- What if there are more than a single or dual outcomes? What if there are a series of different outcomes that are possible?
- For example – calculating your grade

If you have a grade of 80 up to 100 then you receive an A
If you have a grade of 70 to 80 then you receive a B (not including 80)
If you have a grade of 60 to 70 then you receive a C (not including 70)
If you have a grade of 50 to 60 then you receive a D (not including 50)
If you are below a 50 you receive an F

Multiple Outcomes

- Multiple conditions are represented with if followed by else if
- There can be MANY conditions to check
- The first condition is marked with `if`
- The end condition is marked with `else`
- All other conditions are marked with `elif`

Detecting Ranges

- Order of your conditional statements will allow for you to check to see if a value is within a specified range.
  - Each expression indicates the upper range
  - If you fall down to the next condition, and that evaluates to true, then you must be within a specified range

Multiple Outcomes

```
3   grade = 55
4
5   if grade >= 80:
6       print("You have received an A")
7   elif grade >= 70:
8       print("You have received a B")
9   elif grade >= 60:
10      print("You have received a C")
11  elif grade >= 50:
12      print("You have received a D")
13  else:
14      print("You have received an F")
15
16
```

Each branch is preceded by a conditional check

Range check – grade must fall between 79 and 70 if this condition is met

Indentation

Describing complex conditions

- Often two or more conditions are involved in a decision. When this occurs, you have to describe the ***relationship between*** the conditions
- For example
  - A student makes the dean`s list for taking 12 credit hours **AND** having a grade point average of at least 3.5
  - A movie theater offers a discount to anyone who is under 6 years old **OR** over 65 years old
  - An employee gets a bonus vacation day for meeting a sales quota **AND** not being absent for a three month period

Logical operators: and, or and not

- When evaluating two or more conditions, we use different logical operators than the six relational operators we have previously used
- A ***complex condition*** occurs when two or more conditions have to be evaluated for an action to take place
- For example:
  - An employee is eligible for a discount on store items after working two months AND having a perfect attendance record

Complex conditions

- Conditions are joined with the keywords ***and*** or the keyword ***or*** **,**or a single condition can be negated with the use of the keyword ***not***
- The three keywords are known as **logical operators**
- These logical operators are represented as follows:
- `and` - this means both conditions have to be true
- `or` - this means one of the conditions have to be true
- `not` – means the opposite – is true if the value is false

Truth tables

- There are a variety of tools that programmers can use to help them sort out complex logical situations and to make coding easier – truth tables, decision tables, and binary trees
- A truth table is a tool for expressing the results of combinations of conditions

Results for an `and` condition

Condition 1	Condition 2	Condition 1 And Condition 2
T	T	T
T	F	F
F	T	F
F	F	F

Truth tables

- For the Or and Not operator

Results for an `OR` condition

Condition 1	Condition 2	Condition 1 Or Condition 2
T	T	T
T	F	T
F	T	T
F	F	F

Results of a `NOT` condition

Condition 1	Not Condition 1
T	F
F	T

Decision tables

- Some problems are more complex than combination of two conditions
  - Multiple outcomes might be possible
- Decision tables
  - State all relevant conditions
  - True and false combinations of these conditions
  - Outcomes associated with each combination
- The number of combinations is 2 raised to the power of the number of conditions
- For example, if there are 2 conditions the number of possible outcomes is 2^2 or 4 (TT, TF, FT, FF)
- With 3 conditions the number of possible outcomes is 2^3 or 8 (TTT, TTF, TFT, FTT, FTF, TFF, FFF, FFT)

Decision tables

- Decision table structure
  - Made up of rows and columns
  - Number of combinations of conditions
    - 2 raised to the power of the number of conditions
  - Top section has a row for each condition
    - True or false
  - Bottom section has a row for each outcome
    - Performed or not performed
  - Each column after first represents unique combination of condition results

Decision tables

- Imagine a customer that is applying to a bank for a loan. The decision to approve the loan is based on three factors:
  - Whether the customer's income is $40,000 or more
  - Whether the customer has a credit score of 600 or more
  - Whether the customer been employed for longer than a year (12 months)

- This is the bank's policy on loan applications:
  - If the customer's income is $40,000 or more, the credit score is less than 600 and the customer has the same job for more than 12 months – **the loan is approved**
  - If the income is $40,000 or more, the credit score is 600 or more - **the loan is approved**
  - If the income is less than $40,000, the credit score is greater than 600 and the customer has the same job for more than 12 months – **the loan is approved**
  - **Any other combination of factors means the loan is NOT approved**

Decision Tables

- Mark each condition for the column with T or F
- Mark X for each true outcome
- Leave cell blank for each false outcome
- Example: conditions for bank loan

Remember -
for 3 conditions the # of possible outcomes is 2^3 or 8
(TTT, TTF, TFT, FTT, FTF, TFF, FFF, FFT)

Conditions								
Income >= 40000?	T	T	T	T	F	F	F	F
Credit score >= 600?	T	T	F	F	T	T	F	F
Months at job > 12?	T	F	T	F	T	F	T	F
Outcomes								
Approve loan?								

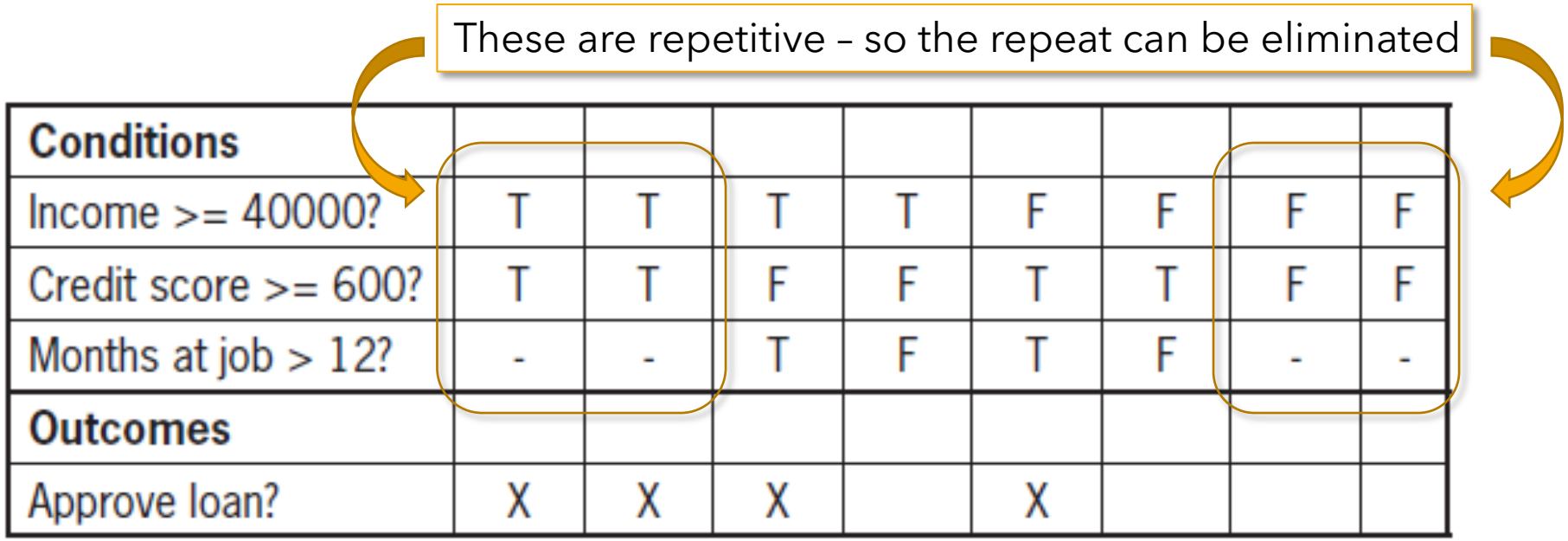
Decision Table

Conditions								
Income >= 40000?	T	T	T	T	F	F	F	F
Credit score >= 600?	T	T	F	F	T	T	F	F
Months at job > 12?	T	F	T	F	T	F	T	F
Outcomes								
Approve loan?	X	X	X		X			

Checking for TRUE conditions

Decision tables

- Can have irrelevant conditions
  - Mark with a dash



Conditions								
Income >= 40000?	T	T	T	T	F	F	F	F
Credit score >= 600?	T	T	F	F	T	T	F	F
Months at job > 12?	-	-	T	F	T	F	-	-
Outcomes								
Approve loan?	X	X	X		X			

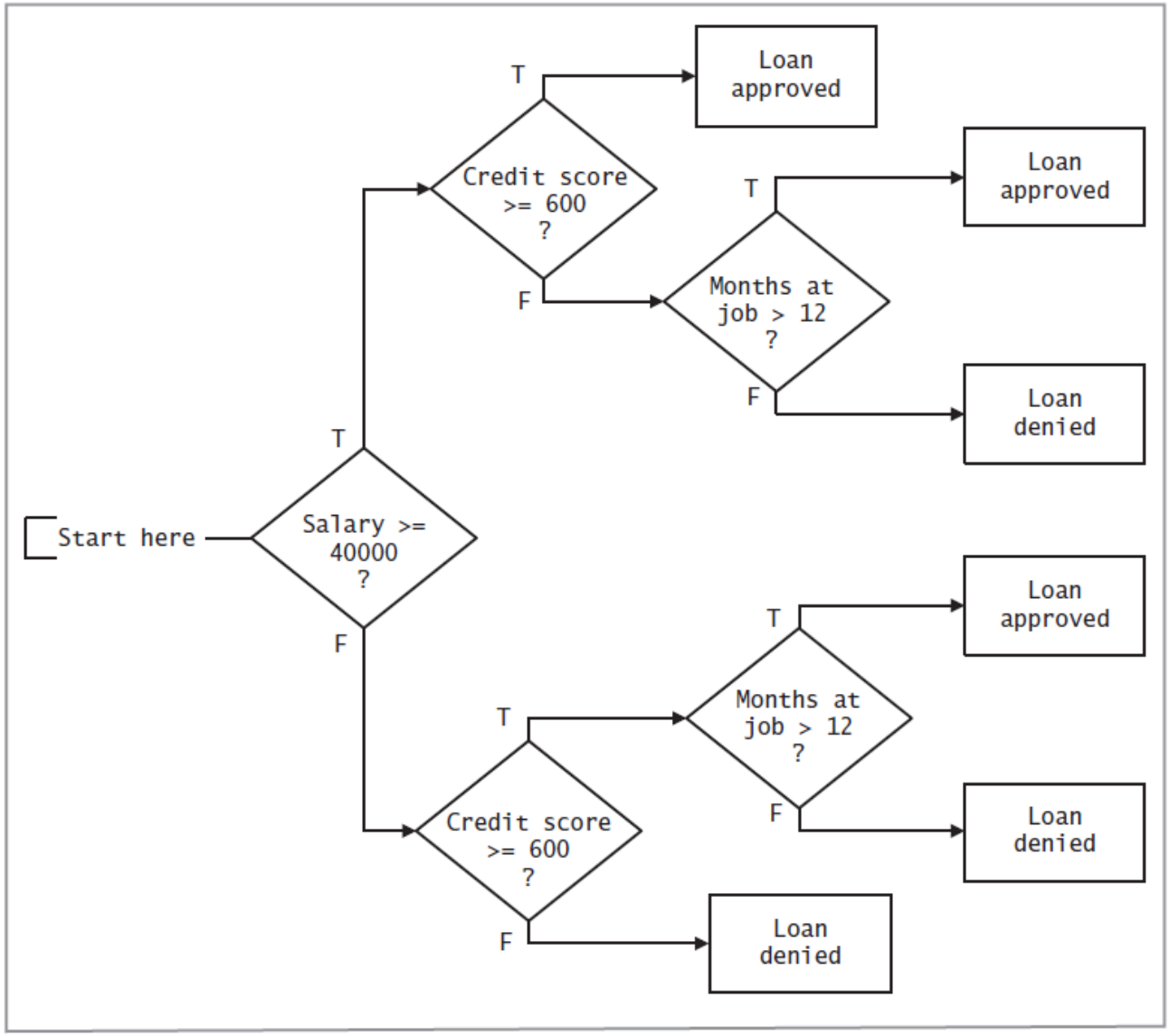
Decision table

- Combine irrelevant condition columns into one

Conditions						
Income >= 40000?	T	T	T	F	F	F
Credit score >= 600?	T	F	F	T	T	F
Months at job > 12?	-	T	F	T	F	-
Outcomes						
Approve loan?	X	X		X		

Binary trees

- Binary trees
  - Trace all combinations
  - Start with one condition
    - Splits into true and false paths
    - Paths lead to next condition
  - Irrelevant conditions don’t split into true and false paths
  - Flowchart shapes often used to draw


Start here
Salary >= 40000 ?
T
F
Credit score >= 600 ?
T
F
Loan approved
Months at job > 12 ?
T
F
Loan approved
Loan denied
Credit score >= 600 ?
T
F
Months at job > 12 ?
T
F
Loan approved
Loan denied
Loan denied

Resulting Code

- Here is the decision table in code

```
3  income = 56000
4  creditScore = 800
5  monthsEmployed = 7
6
7  if income >= 40000 and creditScore >= 600:
8      print("You have been approved")
9  elif income >= 40000 and monthsEmployed > 12:
10     print("You have been approved")
11 elif creditScore >= 600 and monthsEmployed > 12:
12     print("You have been approved")
13 else:
14     print("You have not been approved")
15
```

Nested conditional statements

- A branch's statement can hold additional if-else statements
  - The code is known as a **nested** statement
  - The same structure and conditions apply
    - Indentation
    - If and else and elif end in :

Membership Operators

- Membership operators return a Boolean value – true or false
- They determine if a specified value can be found **in** a container type (using **in**)
  - Or if the value is **not** found in the container (using **not in)**

Membership Operator

```
3
4   students = ["Mal","Jayne", "Washburn", "Zoe"]
5   name = input("Please enter a name:")
6
7   if name in students:
8       print("Welcome - you are registered")
9   else:
10      print("Welcome - you are not on our class list")
11
```

Membership Operators

- Can check any container type
  - Can return if a substring is present in a string
    - For example – if the substring ABCD is in the string “123ABCD”
  - Can return if a key is in a dictionary – does not get the value

Identity Operator

- The identify operator (is) can check of two operands are bound to a single object.
  - Or the inverse using is not
  - They do NOT compare values – they look to see if the two share the same memory address
  - They are not the same as equivalence

Identity Operator

```
3   firstName = "Edgar"
4   lastName = "Smith"
5
6   anotherName = firstName
7
8   if firstName is anotherName:
9       print("Same object")
10  else:
11      print("Different object")
12
```

Same object

Order of Evaluation

Operator/Convention	Description	Explanation
()	Items within parentheses are evaluated first	In `(a * (b + c)) - d`, the + is evaluated first, then *, then -.
* / % + -	Arithmetic operators (using their precedence rules; see earlier section)	`z - 45 * y < 53` evaluates * first, then -, then <.
< <= > >= == !=	Relational, (in)equality, and membership operators	`x < 2 or x >= 10` is evaluated as `(x < 2) or (x >= 10)` because < and >= have precedence over or.
not	not (logical NOT)	`not x or y` is evaluated as `(not x) or y`.
and	Logical AND	`x == 5 or y == 10 and z != 10` is evaluated as `(x == 5) or ((y == 10) and (z != 10))` because and has precedence over or.
or	Logical OR	`x == 7 or x < 2` is evaluated as `(x == 7) or (x < 2)` because < and == have precedence over or.

Ternary Operation

- A ternary operation is a conditional expression with three operands
- Also known as a conditional expression
- Difficult to read, should only be used for simple assignments

Ternary Operation

- If someone has a grade above 50, they passed, otherwise they failed

```
grade = 50

result = "You have passed" if grade >= 50 else "You have failed"

print(result)
```