CONDITIONAL EXECUTION


```
                    ┌──────────┐
                    │          │
        yes         │  x > y?  │      no
       ┌────────────┤          ├────────────┐
       │            │          │            │
       ▼            └──────────┘            ▼
┌──────────────┐              ┌──────────────┐
│   max = x;   │              │   max = y;   │
└──────────────┘              └──────────────┘
```

logical AND	logical OR	logical NOT
and	or	not

Fundamentals of Computer Science I

Outline

- Conditional Execution
  - if …
  - if … else …
  - if … elif … else …
  - Nested if … statements
- A look at Enumerations

Comparisons

- Given two numbers → return a `boolean`

operator	meaning	true example	false example
==	equal	7 == 7	7 == 8
!=	not equal	7 != 8	7 != 7
<	less than	7 < 8	8 < 7
<=	less than or equal	7 <= 7	8 <= 7
>	greater than	8 > 7	7 > 8
>=	greater than or equal	8 >= 2	8 >= 10

Is the sum of a, b and c equal to 0?   `(a + b + c) == 0`
Is grade in the B range?   `(grade >= 80.0) and (grade < 90.0)`
Is `sumItems` an even number?   `(sumItems % 2) == 0`

Leap Year Example

- Years divisible by 4 but not by 100 → leap year
- Years divisible by 400 → leap year

```python
year = int(input("Enter the year: "))
isLeapYear = False

# Leap year if divisible by 4 but not by 100
isLeapYear = (year % 4 == 0) and (year % 100 != 0)

# But also leap year if divisible by 400
isLeapYear = isLeapYear or (year % 400 == 0)
print(isLeapYear)
```

Sequential Flow of Control

time 0	
time 1	
time 2	
time 3	
time 4	
time 5	
time 6	
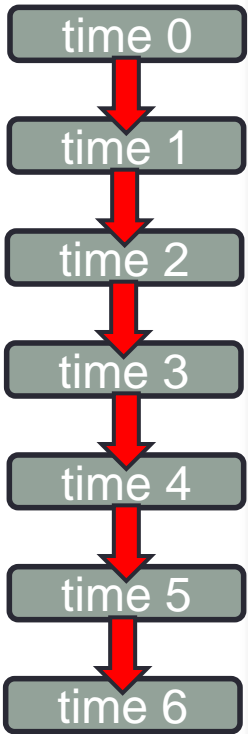
```python
import sys

product = sys.argv[1]

qty     = int(sys.argv[2])

cost    = float(sys.argv[3])

total   = qty * cost

print("To buy ", qty, end = " ")

print(product, end = " ")

print("you will need $" + str(total))
```
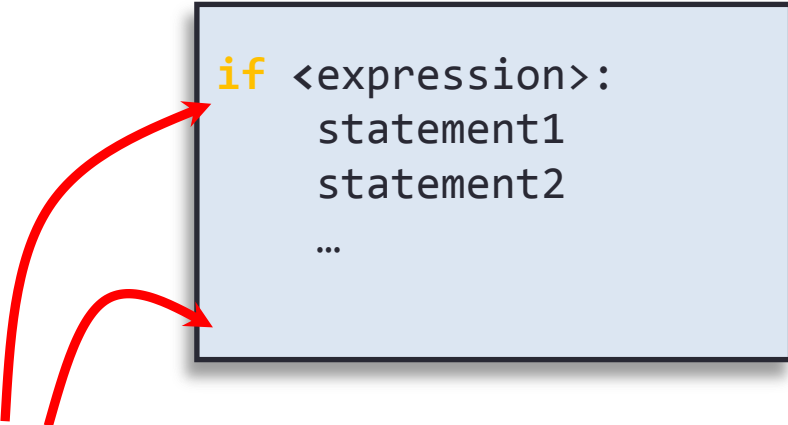
Control Flow

- Default flow of control is sequential

- Interesting and powerful programs need:
  - To skip over some lines
  - To repeat lines

- Conditionals → sometimes skip parts
  - A *branching statement* chooses between two or more possible actions

- Loops → allow repetition of lines
  - A *loop statement* repeats an action until a stopping condition occurs
  - We have talked a bit about **for** loops, and we will talk about loops more

if Statement

- Most common branching statement in all languages
  - Evaluate a `boolean` expression, after the **if**
  - If **True**, do some stuff
  - [optional] If **False**, do some other stuff

```
if <expression>:
    statement1
    statement2
    …
```
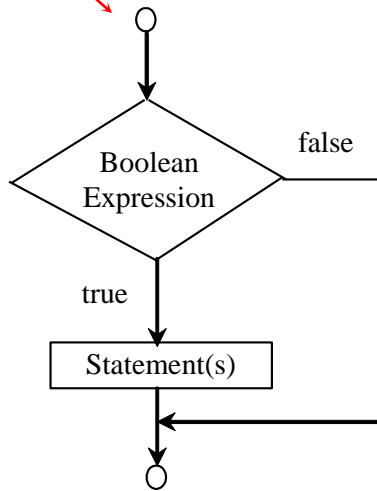
```
if <expression>:
    statement1
    statement2

    …
else:
    statement3
    statement4

    …
```

Indentation used to denote a code "block":
All lines in block get executed (in sequence) or none of the them do
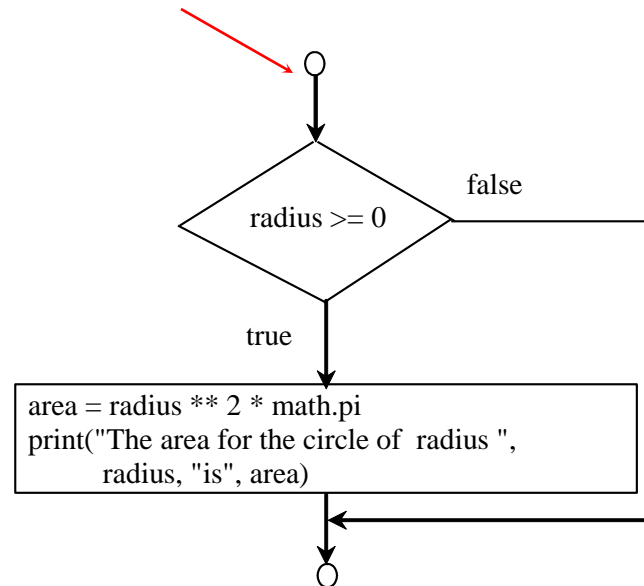
One-way `if` Statements

if <boolean-expression>:
    statement(s)

```
import math
radius = 2
if radius >= 0:
    area = radius ** 2 * math.pi
    print("The area for the circle of radius",
            radius, "is", area)
```
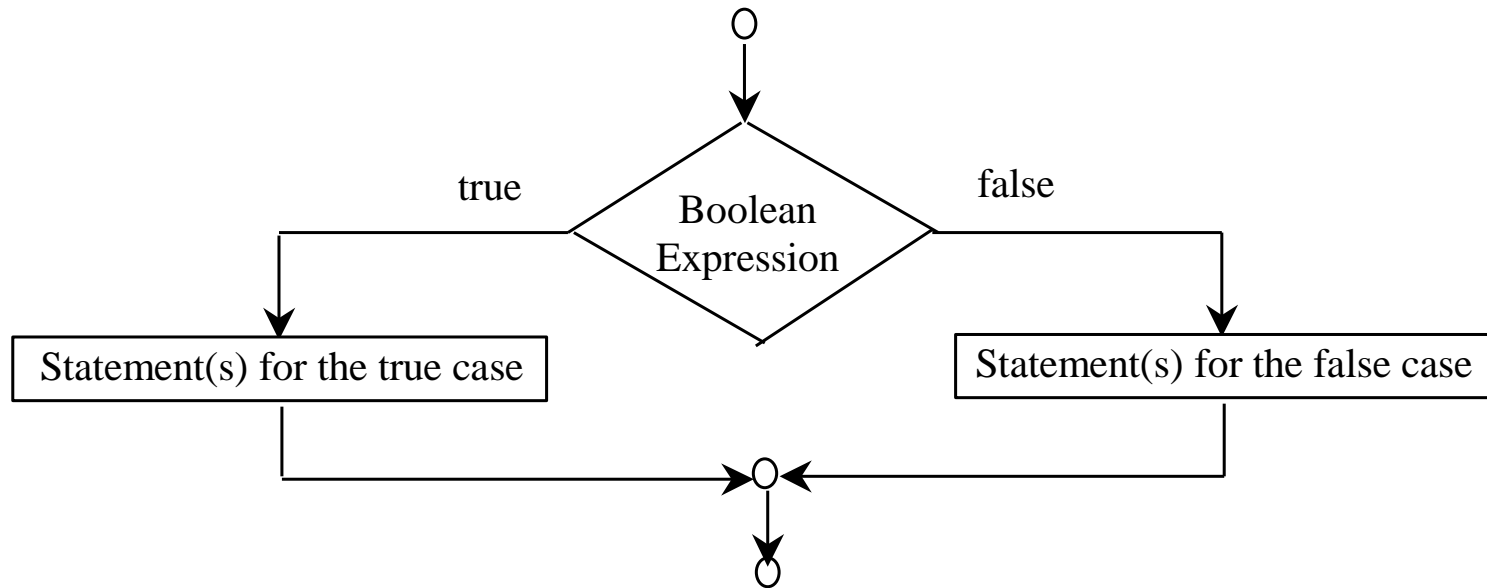

(A)

(B)

The Two-way `if` Statement

```
if <boolean-expression>:
    statement(s)-for-the-true-case
else:
    statement(s)-for-the-false-case
```

if...else Example

```python
import math

radius = -1

if radius >= 0:
    area = radius ** 2 * math.pi
    print("The area for the circle of radius", radius, "is", area)
else:
    print("Negative input")
```

Multiple Alternative if Statements

```python
score = float(input("Enter the score: "))

if score >= 90:
    grade = 'A'
elif score >= 80:
    grade = 'B'
elif score >= 70:
    grade = 'C'
elif score >= 60:
    grade = 'D'
else:
    grade = 'F'

print (grade)
```

Note

The <u>else</u> clause matches the <u>if</u> clause that is at the same level of indentation

```python
if 7 < 5:
    print("Wrong")
    if 5 > 7:
        print("really wrong")
else:
    print("else goes with outer if")
```

TIP

If you have a boolean variable, you can test its value directly instead of testing to see if it is equal to True or False.

```python
even = True

if even == True:
    print("TRUE")

if even:
    print("TRUE again")
```

Booleans

not a   → "Is a set to `false`?"
a and b → "Are both a *and* b set to `true`?"
a or b   → "Is either a *or* b (or both) set to `true`?"

a	b	a and b	a or b
false	false	**false**	**false**
false	true	**false**	**true**
true	false	**false**	**true**
true	true	**true**	**true**

a	not a
true	**false**
false	**true**

if Examples

```
if x < 0:
    x = -x
```

Take absolute value of x

```
import random
if random.random() < 0.5:
    print("heads")
else:
    print("tails")
```

Flip a fair coin and print out the results.

```
if x > y:
    t = x
    x = y
    y = t
```

Put x and y into sorted order

```
if len(sys.argv) > 1:
    num = int(sys.argv[1])
```

If a command line option is passed in, use it as the value for num.

Let's Try One!!

- If the unicorn is mythical, then it is immortal, but if it is not mythical, then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned. The unicorn is magical if it is horned.

- Write a program named **_Unicorn.py_** that determines whether unicorns are mythical, immortal, mortal, mammal, magical or horned, and output these to the screen. Notice, there is no user input to this program. Your program only needs to use the above statements to determine which characteristics are true.

The Ternary Conditional Operator

```
if n1 > n2:

    max = n1

 else:

    max = n2
```

- Can be written as

```
max = (n1 if n1 > n2 else max = n2)
```

- A shortcut for the full if…else statement
- Should only be used for very short pieces of code

The Ternary Conditional Operator

- Useful with print statements.

```python
print("You worked", hours, "hours. " if hours > 1 else "hour.")
```

Input Validation

- You should check user input to ensure that it is within a valid or reasonable range. For example, consider a program that converts feet to inches. You might write the following:

```python
feet = int(input("Enter feet: "))
inches = feet * 12
```

- What if:
  - The user types a negative number for feet?
  - The user enters an unreasonable value like 100? Or a number larger than can be stored in an int? (2,147,483,647)

Input Validation

- Address these problems by ensuring that the entered values are reasonable:

```
feet = int(input("Enter feet: "))

if feet >= 0 and feet < 10:
    inches = feet * 12
    ...
```

Mathematical Expressions: Parentheses and Precedence

- Parentheses can change the order in which arithmetic operations are performed
  - examples:

    ```
    (cost + tax) * discount
    (cost + (tax * discount))
    ```

- Without parentheses, an expressions is evaluated according to the *rules of precedence*, with the lowest precedence listed at the top.

Operator	Description
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
<, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
+, -	Addition and subtraction
*, /, %	Multiplication, division, remainder
**	Exponentiation

Precedence Rules

- In what order are the operations performed?

```
score < min/2 - 10 or score > 90
score < (min/2) - 10 or score > 90
score < ((min/2) - 10) or score > 90
(score < ((min/2) - 10)) or score > 90
(score < ((min/2) - 10)) or (score > 90)
(score < ((min/2) - 10)) or (score > 90))

score < (min/2 – 10 or score > 90)
```

Enumerations

- Consider a need to restrict contents of a variable to certain values
- An enumeration lists the values a variable can have
- Example

```python
import enum
class MovieRating(enum.Enum):
    E = enum.auto()
    A = enum.auto()
    B = enum.auto()

rating = MovieRating.A
```

```python
import enum

class Month(enum.Enum):
    JANUARY = enum.auto()
    FEBRUARY = enum.auto()
    MARCH = enum.auto()
    APRIL = enum.auto()
    MAY = enum.auto()
    JUNE = enum.auto()
    JULY = enum.auto()
    AUGUST = enum.auto()
    SEPTEMBER = enum.auto()
    OCTOBER = enum.auto()
    NOVEMBER = enum.auto()
    DECEMBER = enum.auto()

month = Month.SEPTEMBER
numDays = 0;


if month in [Month.JANUARY, Month.MARCH, Month.MAY, Month.JULY, Month.AUGUST,
Month.OCTOBER, Month.DECEMBER]:
    numDays = 31
elif month in [Month.APRIL, Month.JUNE, Month.SEPTEMBER, Month.NOVEMBER]:
    numDays = 30
elif month in [Month.FEBRUARY]:
    numDays = 28
else:
    print("Invalid month")


print(str(month.name) + " has " + str(numDays) + " days.")
```

Summary

- Conditional Execution
  - if …
  - if … else …
  - if … elif … else …
  - Nested if … statements
- A look at Enumerations

You Try It

- Write a program with an if statement that reports whether you passed or failed based on the character entered at the command line. A, B, and C are passing, D and F are not.

- Submit your program, named Grades.py, to the Activity03 dropbox on Moodle. You get 1 extra credit point for turning something in, 2 points if it is correct.

- Don't forget! Always put your name and a description in a header comment!