CONDITIONAL EXECUTION

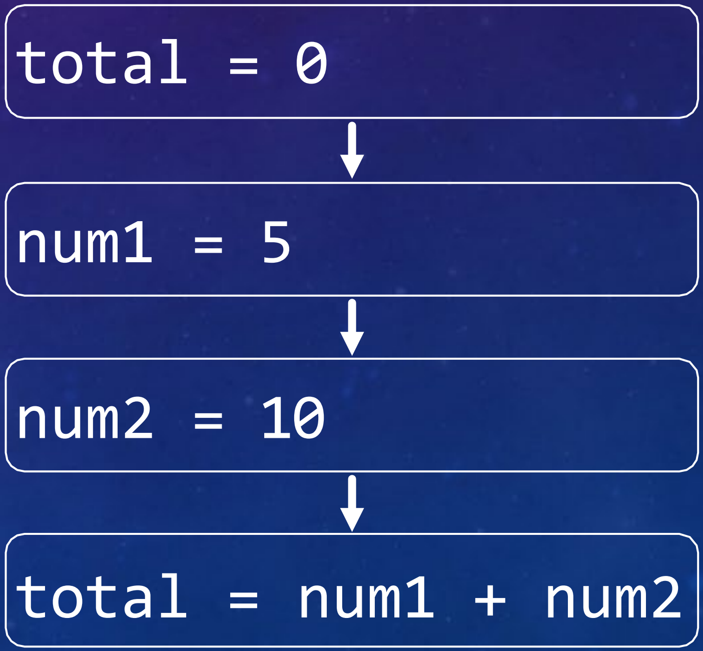
Flow of Control

- Flow of control = order in which statements are executed
- By default, a program's statements are executed sequentially, from top to bottom

Program

```
total = 0
num1 = 5
num2 = 10
total = num1 + num2
```

Flowchart



Conditional Execution

- To solve many types of problems we need to change the standard flow of control
- Conditional execution allows you to decide whether to do something, based on some condition
- Example

```
if x < 0:
    x = -1 * x
```

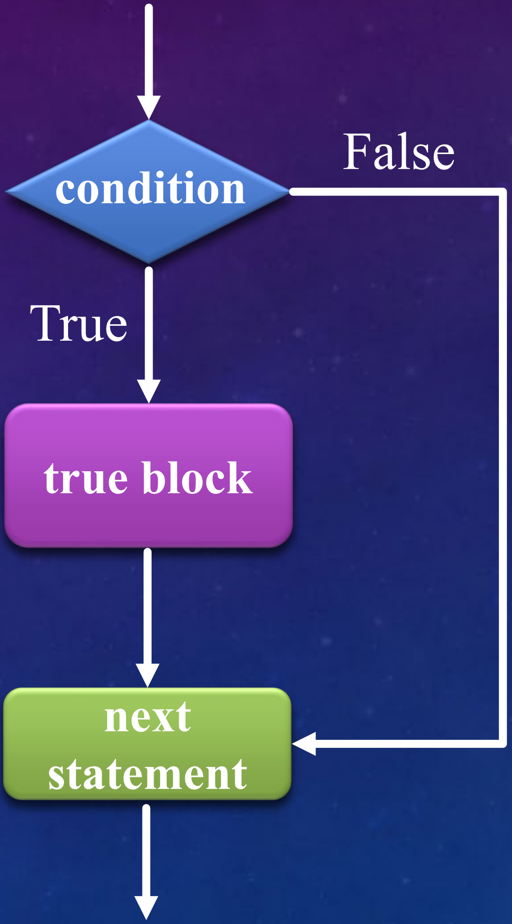
Simple Decisions: *if* Statements

- Syntax:

```
if condition:
    true block
```

where

- Condition is an expression that is true of false
- True block is one or more indented statements



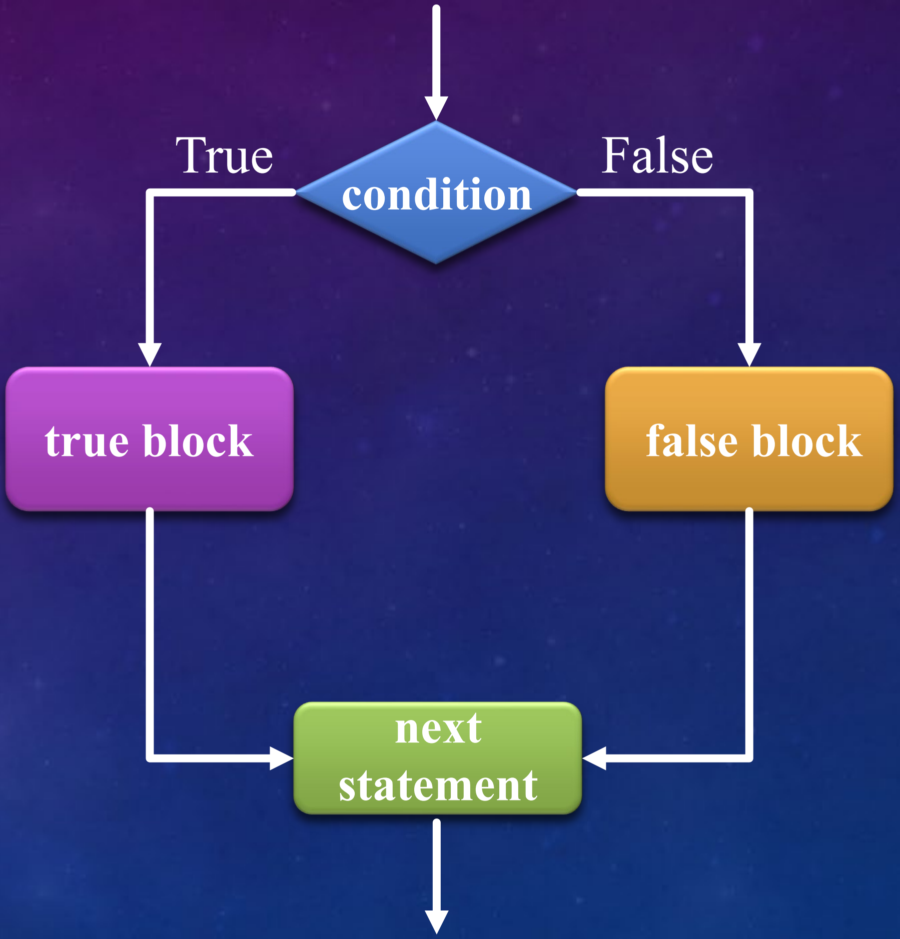
Two-way Decisions: *if-else* Statements

- Syntax:

```
if condition:
    true block
else:
    false block
```

- Example

```
if avg >= 60:
    grade = ‘pass’
else:
    grade = ‘fail’
```



A Word about Blocks

- A block can contain multiple statements

```
if year == ‘frosh’:
    print(‘Welcome to FCU!’)
    print(‘Have a great four years!’)
else:
    print(‘Welcome back!’)
    print(‘Have a great semester!’)
    print(‘Be nice to the frosh students’)
```

- A new block *begins* whenever we increase the amount of indenting
- A block *ends* when we either:
    - Reach a line with less indenting than the start of the block
    - Reach the end of the program

Expressing Simple Conditions

- Python provides a set of *relational operators* for making comparisons

Operators	Name	Examples

`<`	Less than	`val < 10`
		`price < 10.99`

`>`	Greater than	`num > 60`
		`state > ‘ohio’`

`<=`	Less than or equal to	`average <= 85.8`
`>=`	Greater than or equal to	`name >= ‘Jones’`

`==`	Equal to
	Don’t confuse ‘==’ with ‘=’
		`total == 10`
		`letter == ‘P’`

`!=`	Not equal to	`age != my_age`

Boolean Expressions

- A condition has one of two values: True or False

```
print(10 < 20)
True
print(10 < 20 < 15)
False
print('Jones' == 'Baker')
False
```

- True and False are NOT strings
  - They are literals form the bool data type

```
print(type(True))
<class 'bool'>
print(type(30 > 6))
<class 'bool'>
```

- An expression that evaluates to True or False is known as a *Boolean expression*

Forming More Complex Conditions

- Python provides *logical operators* for combining/modifying Boolean expressions

Operators	Example and Meaning

`and`	`age >= 18 and age <= 35`
	`True if both conditions are True`
	`False otherwise`

`or`	`age < 3 or age > 65`
	`True if one or both of the conditions are Ture`
	`False if both conditions are False`

`not`	`not (grade > 80)`
	`True if the condition is False`
	`False if it is True`

Nesting

- We can “nest” one conditional statement in the true block or false block of another conditional statement

```
if year == ‘fresh’:
    print(‘Welcome to FCU!’)
    print(‘Have a great four years!’)
else:
    print(‘Welcome back!’)
    if year == ‘senior’:
        print(‘Have a great last year!’)
    else:
        print(‘Have a great semester!’)
    print(‘Be nice to the frosh students’)
```

What is the Output of the Following Program?

```
x = 5
if x < 15:  # true
    ~~if x > 8:~~  # false
        ~~print('one')~~
    else:
        print('two')
else:
    ~~if x > 2:~~
        ~~print('three')~~
```

A. one
B. two
C. three
D. More than one of the above
E. Nothing is output

What is the Output of the Following Program?

```
x = 5
if x < 15:  # true
    ~~if x > 8:~~  # false
        ~~print('one')~~
    else:
        print('two')
else:
    ~~if x > 2:~~
        ~~print('three')~~
```

A. one

B. two

C. three

D. More than one of the above

E. Nothing is output

What Does This Print? (Note the Changes!)

```
x = 5
if x < 15: # true
    ~~if x > 8:~~ # false
        ~~print(‘one’)~~
    else:
        print(‘two’)
if x > 2: # true
        print(‘three’)
```

A. one
B. two
C. three
D. More than one of the above
E. Nothing is output

What Does This Print? (Note the Changes!)

```
x = 5
if x < 15: # true
    ~~if x > 8:~~ # false
        ~~print(‘one’)~~
    else:
        print(‘two’)
if x > 2: # true
        print(‘three’)
```

A. one
B. two
C. three
D. More than one of the above
E. Nothing is output

What Does This Print? (Note the New Changes!)

```
x = 5
if x < 15: # true
    ~~if x > 8:~~ # false
        ~~print(‘one’)~~
~~else:~~
    ~~print(‘two’)~~
if x > 2:
        print(‘three’)
```

A. one
B. two
C. three
D. More than one of the above
E. Nothing is output

What Does This Print? (Note the New Changes!)

```
x = 5
if x < 15: # true
    ~~if x > 8:~~# false
        ~~print(‘one’)~~
~~else:~~
    ~~print(‘two’)~~
if x > 2:
        print(‘three’)
```

A. one
B. two
C. three
D. More than one of the above
E. Nothing is output

Multi-way Decisions

- The following code doesn’t work

```
avg = 95
if avg >= 90:
    grade = ‘A’
if avg >= 80:
    grade = ‘B’
if avg >= 70:
    grade = ‘C’
if avg >= 60:
    grade = ‘D’
else:
    grade = ‘F’
```

```
print(grade)

D
```

Multi-way Decisions

- Here's a fixed version

```
avg = 95
if avg >= 90:
    grade = ‘A’
elif avg >= 80:
    grade = ‘B’
elif avg >= 70:
    grade = ‘C’
elif avg >= 60:
    grade = ‘D’
else:
    grade = ‘F’
```

```
print(grade)
```

```
A
```

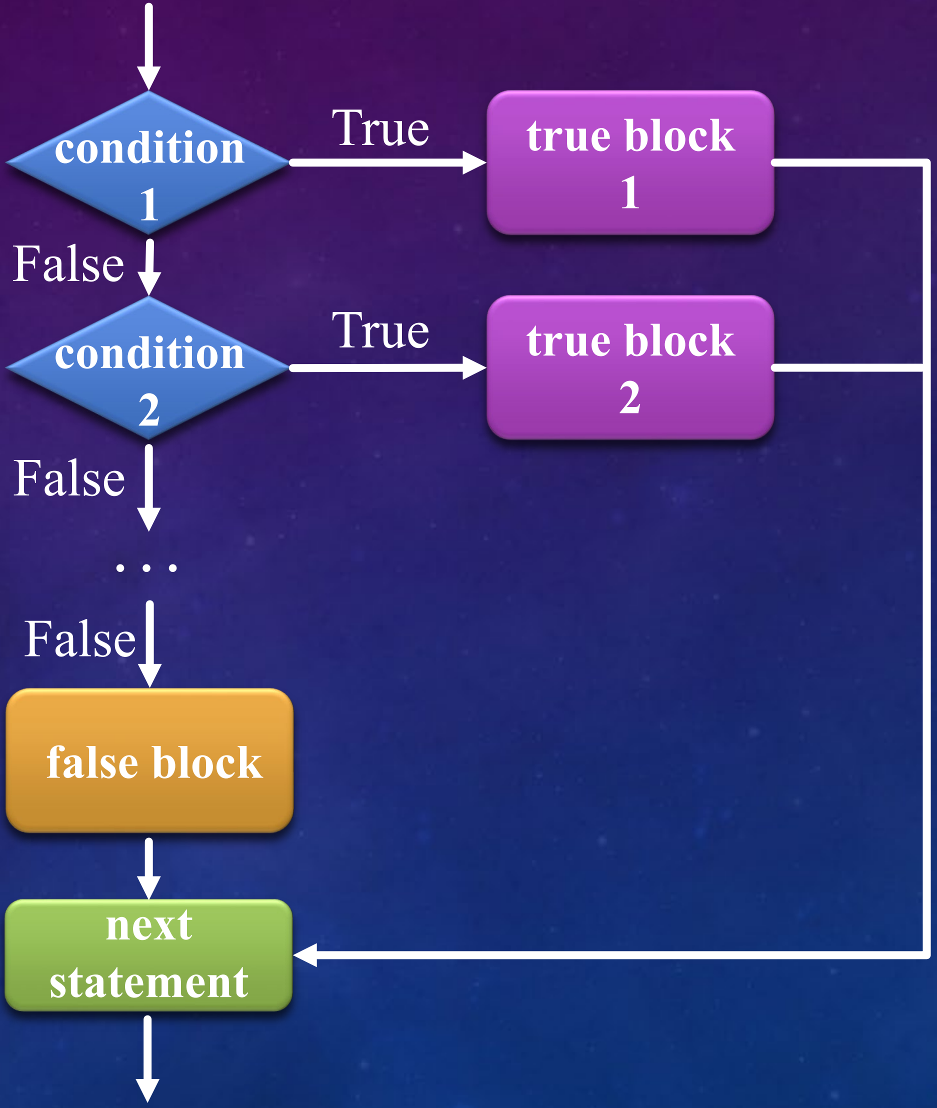
Multi-way Decisions: *if-elif-else* Statements

- Syntax:

```
if condition1:
    true block for condition1
elif condition2:
    true block for condition2
elif condition3:
    true block for condition3
...
else:
    false block
```

- The conditions are evaluated in order. The true block of the first true condition is executed
- If none of the conditions are true, the false block is executed

Flowchart for an *if-elif-else* Statement



How Many Lines Does This Print?

```
x = 5
if x == 8:
    print(‘how’)
elif x > 1:
    print(‘now’)
elif x < 20:
    print(‘brown’)
print(‘cow’)
```

A. 0
B. 1
C. 2
D. 3
E. 4

How Many Lines Does This Print?

```
x = 5
if x == 8:
    print('how')
elif x > 1:
    print('now')
elif x < 20:
    print('brown')
print('cow')
```

A. 0
B. 1
C. 2
D. 3
E. 4

How Many Lines Does This Print?

```
x = 5
if x == 8:
    print(‘how’)
if x > 1:
    print(‘now’)
if x < 20:
    print(‘brown’)
print(‘cow’)
```

A. 0
B. 1
C. 2
D. 3
E. 4

How Many Lines Does This Print?

```
x = 5
if x == 8:
    print('how')
if x > 1:
    print('now')
if x < 20:
    print('brown')
print('cow')
```

A. 0
B. 1
C. 2
D. 3
E. 4

Common Mistakes When Using and/or

```
if a == 0 or 1:
    y = y + 1
    print(y)
```

← This is problematic!

- When using and/or, both sides of the operator should be a Boolean expression that could stand on its own

```
  boolean        boolean
a == 0 or a == 1:
```

DO THIS

```
boolean   integer
a == 0 or 1:
```

DON'T DO THIS!

- Unfortunately, Python doesn't complain about code like the problematic code above
    - But it won't typically work the way you want it to

Avoid Overly Complicated Code

- The following also involves decisions based on a person's age

```
age = ... # let the user enter his/her age
if age < 13:
    print('You are a child.')
elif ~~age >= 13 and~~ age < 20:
    print('You are a teenager.')
elif ~~age >= 20 and~~ age < 30:
    print('You are in your twenties.')
elif ~~age >= 30 and~~ age < 40:
    print('You are in your thirties.')
else:
    print('You are a survivor.')
```

- How could it be simplified?