Objectives

- More conditionals!
- Boolean operators

1

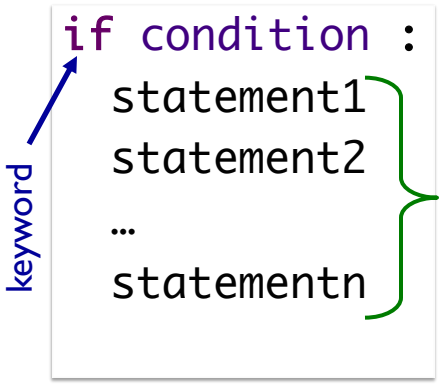
Review

- How can we make Python code execute only under certain circumstances?
- How do we say "otherwise" in Python?
- How do we write the condition that evaluates to True if two expressions (let's say `expr1` and `expr2`) are equal?
  - How do we write the condition to evaluate to True only if those two expressions are *not* equal?

2

Review: Syntax of `if` statement:

Simple Decision



- Note indentation

English Examples:

if it is raining **:**
    I will wear a raincoat

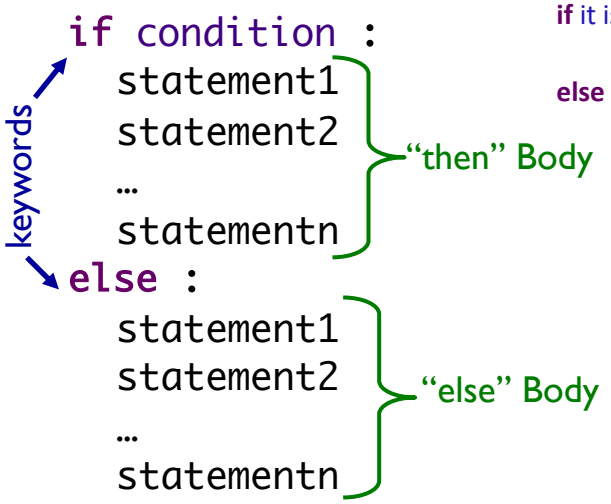
if the PB is new **:**
    Remove the seal

Syntax of `if` statement:

Two-Way Decision

English Example:

if it is Saturday or Sunday :
    I wake up at 9 a.m.
else :
    I wake up at 7 a.m.



Review: Relational Operators

- Syntax:
  - `<expr> <relational_operator> <expr>`

Low precedence

Relational Operator	Meaning
<	Less than?
<=	Less than or equal to?
>	Greater than?
>=	Greater than or equal to?
==	Equals?
!=	Not equals?

Review: Using Conditionals

- Determine if a number is even or odd

```
x = eval(input("Enter a number: "))
remainder = x%2
if remainder == 0:
    print(x, "is even")
if remainder == 1:
    print(x, "is odd")
```

```
x = eval(input("Enter a number: "))
remainder = x % 2
if remainder == 0:
    print(x, "is even")
else:
    print(x, "is odd")
```

This is the more efficient implementation. Why?

Review: Test-Driven Development (TDD)

- Create test cases first
- Idea: Focus on the outcomes first
- Helps you think about the problem without thinking about the code itself

Review: Speeding Ticket Fines

- Any speed clocked over the limit results in a fine of at least $50, plus $5 for each mph over the limit, plus a penalty of $200 for any speed over 90mph.

- Our function
  - Input: speed limit and the clocked speed
  - Output: the appropriate fine
    - What should the appropriate fine be if the user is not speeding?

`speedingticket.py`

Speeding Ticket Fine

```
def calculateFine( speed, speedLimit ):
    """
    Calculates the fine (explain…)
    Precondition: speed and speedlimit are both non-
    negative integers
    Returns 0 if not speeding; otherwise, returns the fine
    """

    if speed <= speedLimit:
        return 0
    else:
        # calculate the fine
        mphOver = speed – speedLimit
        fine = 50 + mphOver * 5

        # excessive speed
        if speed > 90:
            fine = fine + 200

        return fine
```

Other correct implementations too

9

Practice: Speeding Ticket Fines

- Any speed clocked over the limit results in a fine of at least $50, plus $5 for each mph over the limit, plus a penalty of $200 for any speed over 90mph.

- Our **program**
  - Input: speed limit and the clocked speed
  - Output: appropriate output to the user, *based on their speeding/fine*

`speedingticket.py`

10

Practice: Speeding Ticket Fines

```
def main():
    print("This program …")

    clockedSpeed = eval(input("Enter your speed: "))
    speedLimit = eval(input("Enter the speed limit: "))

    # your code here

def calculateFine(limit, speed):
    …
```

- Our **program**
  - Input: speed limit and the clocked speed
  - Output: appropriate output to the user, *based on their speeding/fine*

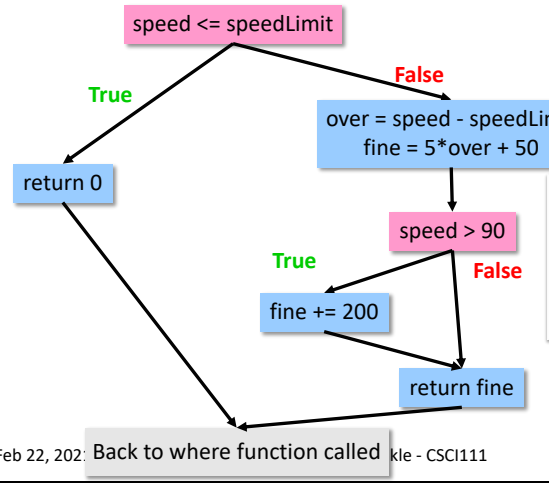
speedingticket.py

Testing Speeding Ticket Program

- Our test cases fell into two categories:
  - Data-related
    - Make sure we picked good numbers (clocked speed: 90, 91)
    - Consider ***boundary*** conditions
  - Control-related
    - Make sure we're hitting all the possible control-related cases, e.g., not speeding, speeding, excessive speeding

Testing with `if` Statements

- Make sure *at least* have test cases that execute each branch in control flow diagram
  - i.e., Each execution path is "covered"



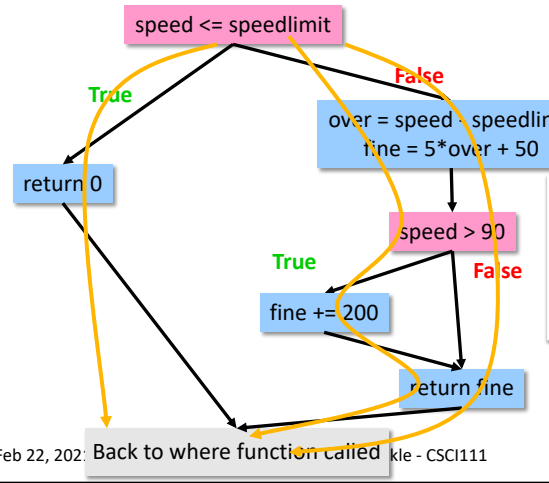
Three execution paths

```
if speed <= speedLimit:
    return 0
else:
    diff = speed - speedLimit
    fine = 50 + 5 * diff
    if speed > 90:
        fine += 200
    return fine
```


13

Testing with `if` Statements

- Make sure *at least* have test cases that execute each branch in control flow diagram
  - i.e., Each execution path is "covered"



Three execution paths

```
if speed <= speedlimit:
    return 0
else:
    diff = speed - speedlimit
    fine = 50 + 5 * diff
    if speed > 90:
        fine += 200
    return fine
```


14

Using the building blocks: Nesting `if-else` statements

```
if condition :
    if condition :
        statements
    else:
        statements
else:
    statements
    if condition :
        statements
    else:
        statements
```

`if-else` statement is **nested** inside the `if`

`if-else` statement is **nested** inside the `else`

Practice: Numeric to Letter Grade

- Write a function to determine a numeric grade's letter grade (A, B, C, D, or F)

Numeric Grade	Letter Grade
90 and above	A
80 to below 90	B
70 to below 80	C
60 to below 70	D
Below 60	F

`grade_function.py`

Syntax of **if** statement: Multi-Way Decision

keywords

```
if condition :
   <then-body1>
elif condition :
   <then-body2>
elif condition :
   <then-body3>
   …
else:
    <default-body>
```

English Example:

if it is Saturday:
    I wake up at 10 a.m.
elif it is Sunday:
    I wake up at 9 a.m.
else:
    I wake up at 7 a.m.

Using the building blocks: Nesting `if-else` statements

```
if condition:
   statements
else:
   if condition:
      statements
   else:
      statements
```

`if-else` statement is nested inside the `else`

This structure can be rewritten as an if-elif-else statement

If-Else-If statements

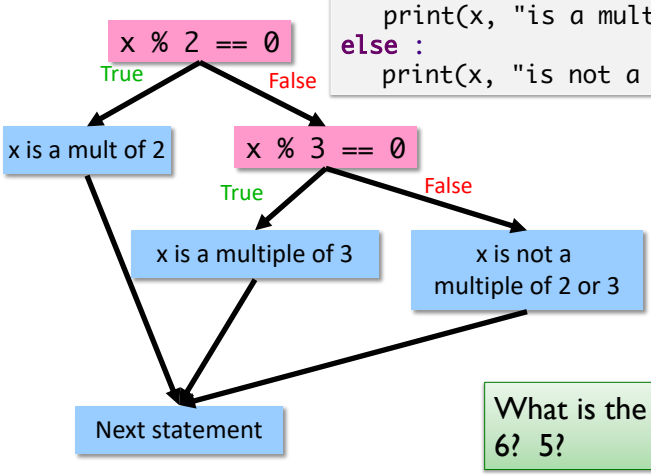
Draw the control flow diagram

```
if x % 2 == 0 :
   print(x, "is a multiple of 2")
elif x % 3 == 0 :
   print(x, "is a multiple of 3")
else :
   print(x, "is not a multiple of 2 or 3")
```


19

If-Else-If statements

```
if x % 2 == 0 :
   print(x, "is a multiple of 2")
elif x % 3 == 0 :
   print(x, "is a multiple of 3")
else :
   print(x, "is not a multiple of 2 or 3")
```

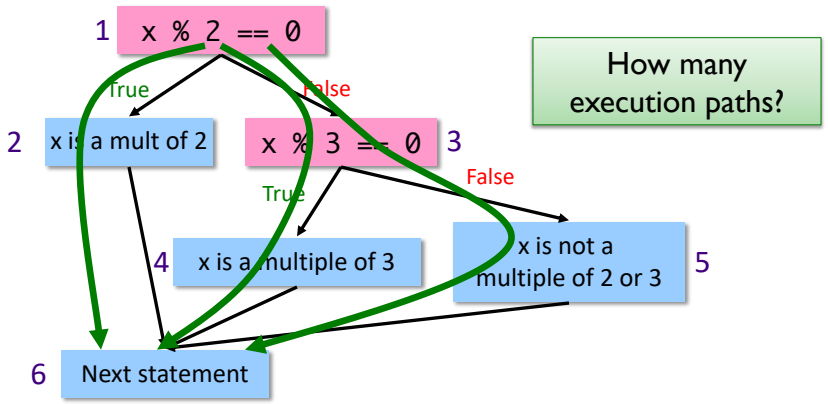


What is the output if x is 4? 6? 5?

20

Testing with If Statements

- Make sure have test cases that execute each branch in control flow diagram
  - i.e., Each execution path is "covered"



Modify to use `elif`

- Determine if a numeric grade is a letter grade (A, B, C, D, or F)

Numeric Grade	Letter Grade
90 and above	A
80 to below 90	B
70 to below 80	C
60 to below 70	D
Below 60	F

More Complex Conditions

- Boolean
  - Two logical values: True and False
- Combine conditions with Boolean operators
  - **and** – True only if **both** operands are True
  - **or** – True if **at least one** operand is True
  - **not** – True if the operand is not True
- English examples
  - If it is raining **and** it is cold
  - If it is Saturday **or** it is Sunday
  - If the shirt is on sale **or** the shirt is purple

What is the output?

```
x = 2
y = 3
z = 4

b = x==2
c = not b
d = (y<4) and (z<3)
print("d=",d)
d = (y<4) or (z<3)
print("d=",d)

d = not d
print(b, c, d)
```

Focus: how operations work
Not good variable names

Because of precedence, we don't *need* parentheses

Truth Tables

operands

A	B	A and B	A or B	not A	not B	not A and B	A or not B
T	T						
T	F						
F	T						
F	F						

25

Truth Tables

operands

A	B	A and B	A or B	not A	not B	not A and B	A or not B
T	T	T	T				
T	F	F	T				
F	T	F	T				
F	F	F	F				

26

Truth Tables

operands

A	B	A and B	A or B	not A	not B	not A and B	A or not B
T	T	T	T	F	F		
T	F	F	T	F	T		
F	T	F	T	T	F		
F	F	F	F	T	T		

27

Truth Tables

operands

A	B	A and B	A or B	not A	not B	not A and B	A or not B
T	T	T	T	F	F	F	T
T	F	F	T	F	T	F	T
F	T	F	T	T	F	T	F
F	F	F	F	T	T	F	T

28

Practice: Numeric Grade Input Range

- Enforce that user must input a numeric grade between 0 and 100
  - In Python, we can't (always) write a condition like 0 <= num_grade <= 100, so we need to break it into two conditions
- Write an appropriate condition for this check on the numeric grade
  - Using **and**
  - Using **or**

Practice: Numeric Grade Input Range

- Enforce that user must input a numeric grade between 0 and 100
  - Using **and**

```
if num_grade >= 0 and num_grade <= 100:
    Computation (call function, etc.)
else:
    print error message
```

  - Using **or**

```
if num_grade < 0 or num_grade > 100:
    print error message
else:
    Computation (call function, etc.)
```

Short-circuit Evaluation

- Don't necessarily need to evaluate all expressions in a compound expression
- A `and` B
  - If A is `False`, compound expression is `False`
- A `or` B
  - If A is `True`, compound expression is `True`
- No need to evaluate B
  - Put more important/limiting expression first
  - Example:

```
if count != 0 and sum/count > 10:
    do something
```


Looking Ahead

- Pre lab 5 due tomorrow, before lab
- Lab 5 tomorrow
- BI: self-driving cars