Objectives

- More Conditionals
- Boolean Operators

Review

- How can we make Python code execute only under certain circumstances?
  - Describe the syntax and semantics
- How do we say “otherwise” in Python?
- What are relational operators?
  - Provide examples
- (From last Wed) Speeding ticket fine function
  - Recall test cases; implement the function

Review: Simple Decision

```
if condition :
    statement1
    statement2
    …
    statementn
```

keyword

"then" Body

- Note indentation

English Examples:

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

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

Review: Two-Way Decision

```
if condition :
    statement1
    statement2
    …
    statementn
else :
    statement1
    statement2
    …
    statementn
```

keywords

“then” Body

“else” Body

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>`
- Evaluates to either `True` or `False`
  - Boolean type

Low precedence
After arithmetic operators

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

Use Python interpreter

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")
```

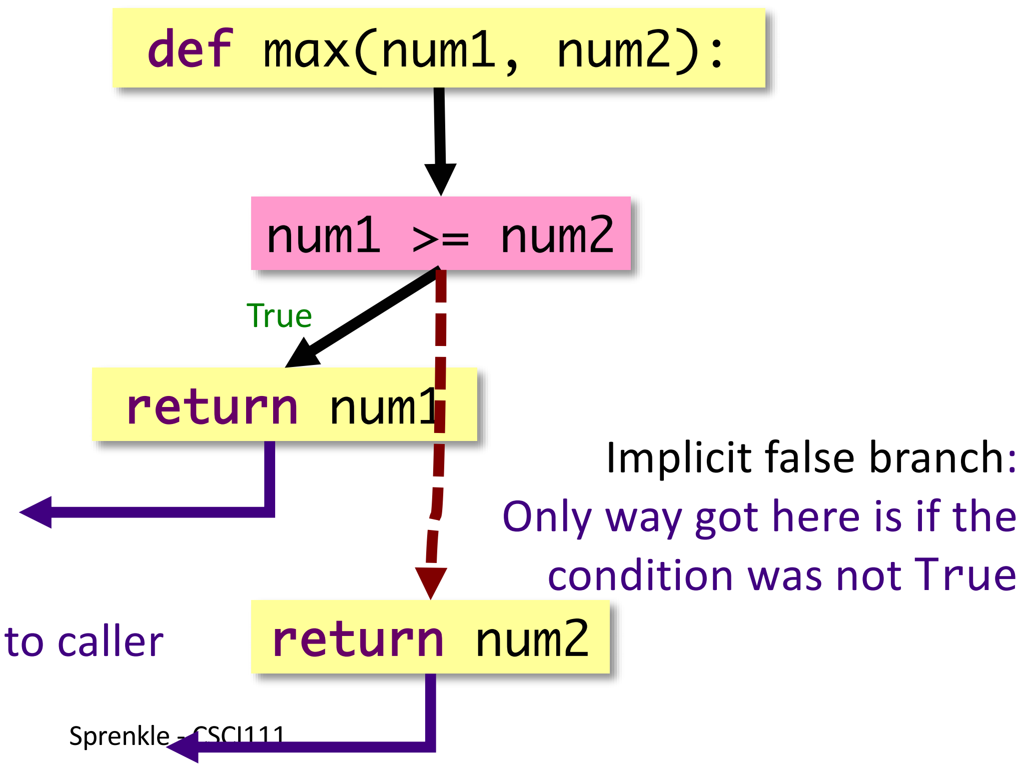
```
x = eval(input("Enter a number: "))
remainder = x % 2
if remainder == 0:
      print(x, "is even")
else:
      print(x, "is odd")
```

This is the more efficient implementation. Why?

Review: Flow of Control: Using `return`

Is this implementation of the function correct? Yes!

```
def max(num1, num2):
    if num1 >= num2:
        return num1
    return num2
```



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

Testing Speeding Ticket Program

- Our test cases fell into two (not mutually exclusive) 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 (below and equal to speed limit), speeding, excessive speeding

`speedingticket.py`

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

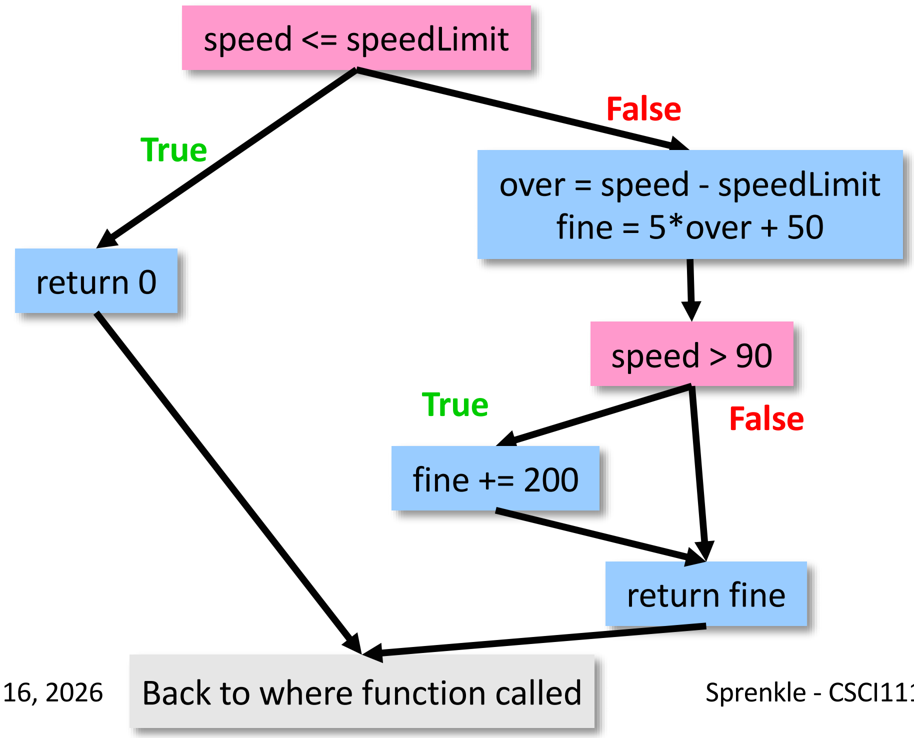
Our process

✓ Write test cases
2. Implement function
3. Test function

`speedingticket.py`

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”

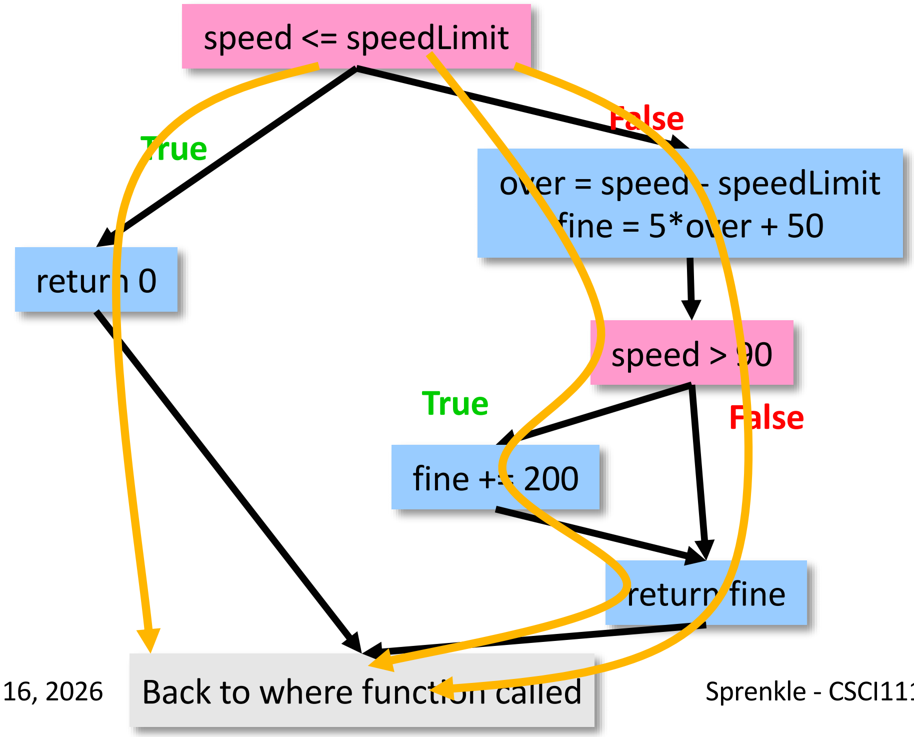


Three execution paths

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

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
```

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`

Practice: Speeding Ticket Fines

- Any speed o
  least $50, p
  penalty of $

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

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

    # your code here

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

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

`speedingticket.py`

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 program 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	

```python
numericGrade = float(input("Numeric grade: "))

# Your code here...

print("Your grade is", letterGrade)
```

grade.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

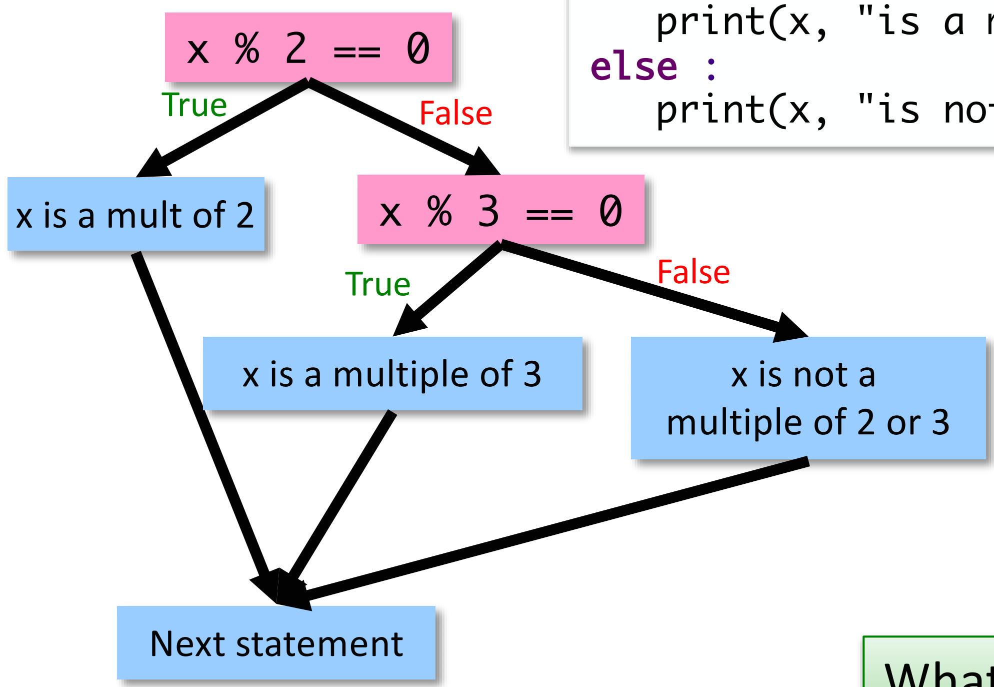
Draw the control flow diagram

```
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")
```

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

If-Else-If statements

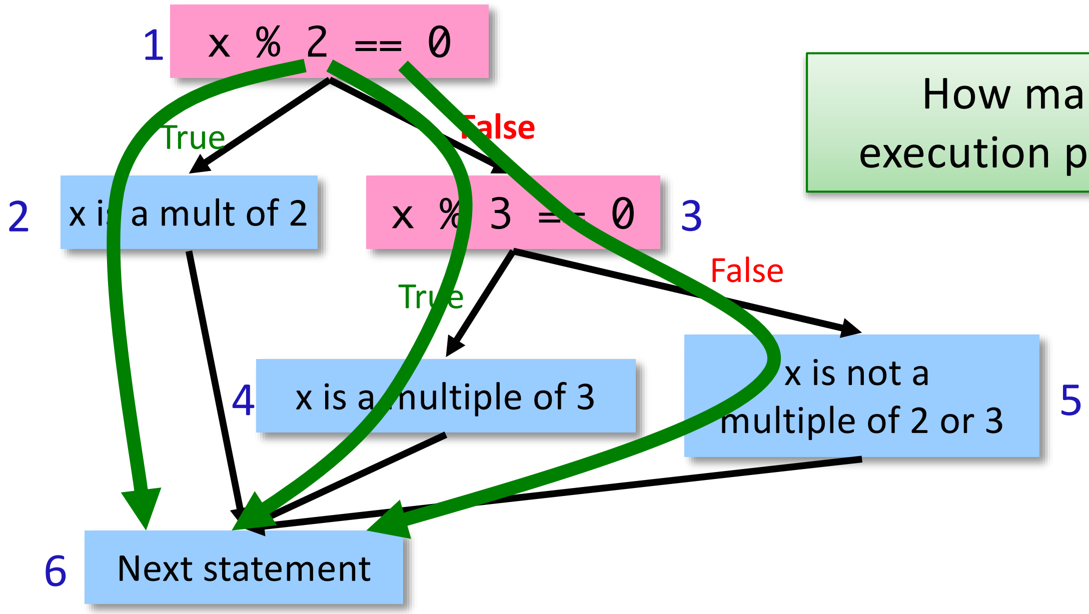
```
if x % 2 == 0 :
   print(x, "is a multiple of 2")
elif x % 3 == 0 :
   print(x, "is a multiple of 3")
else :
   print(x, "is not a multiple of 2 or 3")
```



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

Testing with If Statements

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



How many execution paths?

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

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						

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				

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		

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

What is the output?

Focus: how operations work

These are *not* good variable names

```
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)
```

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

`eval_cond.py`

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**

Focus on the ***condition***
Then, we'll block out the code

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
```

Determining Multiples

multiples.py

Original Code:
Emphasized mutually
exclusive behavior

```
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")
```

If you want different behavior…

```
isNotDivisibleBy2Or3 = True

if x % 2 == 0:
    print(x, "is a multiple of 2")
    isNotDivisibleBy2Or3 = False

if x % 3 == 0:
    print(x, "is a multiple of 3")
    isNotDivisibleBy2Or3 = False

if isNotDivisibleBy2Or3:
    print(x, "is not a multiple of 2 or 3")
```

Compare control flow diagrams

Determining Multiples

multiples.py

Original Code:
Emphasized mutually exclusive behavior

```
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")
```

```
isNotDivisibleBy2Or3 = True

1 if x % 2 == 0:
2     print(x, "is a multiple of 2")
      isNotDivisibleBy2Or3 = False

3 if x % 3 == 0:
4     print(x, "is a multiple of 3")
      isNotDivisibleBy2Or3 = False

5 if isNotDivisibleBy2Or3:
6     print(x, "is not a multiple of 2 or 3")
```

What statements execute when x is 4, 5, 6, and 9?

Looking Ahead

- Pre lab 5 due tomorrow, before lab
- Lab 5 tomorrow
- BI: Autonomous Vehicles