

## Objectives

- Conditionals
- Creating modules
- Exam review

Feb 9, 2022

Sprenkle - CSCI111

1

1

## Review

- What makes a “good” function?
- What are benefits of functions?
- What does it mean to “programmatically test” a function?
  - What are characteristics of functions that we can programmatically test?
- How do we organize programs with functions (so far)?
- What development approaches did we discuss?
  - What are their steps?

Feb 9, 2022

Sprenkle - CSCI111

2

2

## Review: Writing a “Good” Function

- Should be an “intuitive chunk”
  - Doesn’t do too much or too little
  - If does too much, try to break into more functions
- Should be reusable
- Should have an “action” name
- Should have a comment that tells what the function does

Feb 9, 2022

Sprenkle - CSCI111

3

3

## Review: Why Write Functions?

- Allows you to break up a problem into *smaller*, more *manageable* parts
- Makes your code easier to *understand*
- Hides implementation details (*abstraction*)
  - Provides interface (input, output)
- Makes part of the code *reusable* so that you:
  - Only have to write function code once
  - Can debug it all at once
    - Isolates errors
  - Can make changes in one function (*maintainability*)

Feb 9, 2022

Sprenkle - CSCI111

4

4

## Review: Where are Functions Defined?

- Functions can go inside program script
  - If no `main()` function, defined *before* use/called
  - If `main()` function, defined anywhere in script
- Functions can go inside a separate *module*

Feb 9, 2022

Sprenkle - CSCI111

5

5

## Review: Refactoring:

### Converting Functionality into Functions

1. Identify functionality that should be put into a function
  - What should the function do?
  - What is the function's input?
  - What is the function's output (i.e., what is returned)?
2. Define the function
  - Write comments
3. Test the function programmatically
  - Comment out the other code temporarily
4. Call the function where appropriate
5. Create a `main` function that contains the "driver" for your program
  - Put at top of program
6. Call `main` at bottom of program

Feb 9, 2022

Sprenkle - CSCI111

6

6

## Why Refactoring?

- Common practice: write code, then realize it would be better (more readable, reusable, ...) if it were in a function
- For us: helpful to separate the code implementation from the function implementation

Feb 9, 2022

Sprenkle - CSCI111

7

7

## Review: Testing Functions

- Functions make it easier for us to test our code
- We can write code to test the functions
  - Test Case:
    - Input: parameters
    - Expected Output: what we expect to be returned
      - Or if state changed as we expected
  - We can verify the function *programmatically*
    - “programmatically” – automatically execute test cases and verify that the actual returned result is what we expected
    - No user input required!

Feb 9, 2022

Sprenkle - CSCI111

8

8

## Review: test Module

- Not a standard module
  - Included with our textbook
  - More sophisticated testing modules but this is sufficient for us
- Function:
  - `testEqual(actual, expected[, places=5])`
    - Parameters: actual and expected results for a function.
    - Displays "Pass" and returns True if the test case passes.
    - Displays error message, with expected and actual results, and returns False if test case fails.

Feb 9, 2022

Sprenkle - CSCI111

9

9

## test module's testEqual function

```
def testWinPercentage():
    test.testEqual( calculateWinPercentage(0, 1), 0 )
    test.testEqual( calculateWinPercentage(2, 2), .5 )
    test.testEqual( calculateWinPercentage(3, 7), .3 )
    test.testEqual( calculateWinPercentage(1, 0), 1 )
```

`testWinPercentage()` Could add a parameter for the number of decimal places of precision

After confirming that the function works...

Feb 9, 2022

Sprenkle - CSCI111

10

10

## test module's testEqual function

```
def testWinPercentage():
    test.testEqual( calculateWinPercentage(0, 1), 0 )
    test.testEqual( calculateWinPercentage(2, 2), .5 )
    test.testEqual( calculateWinPercentage(3, 7), .3 )
    test.testEqual( calculateWinPercentage(1, 0), 1 )

# testWinPercentage()
main()
```

Comment out call to test function.  
Call main.


## Another Example of Programmatic Testing

- Testing a constructor/function/method that affects state:

```
def testGraphWin():
    window = GraphWin("Title", 300, 200)
    test.testEqual( window.getWidth(), 300 )
    test.testEquals( window.getHeight(), 200 )
    ...
```

- Call the constructor/function/method under test
- Check the resulting state

## Parts of an Algorithm

- Input, Output
  - Primitive operations
    - What data you have, what you can do to the data
  - Naming
    - Identify things we're using
  - Sequence of operations
  - Conditionals
  - Repetition/Loops
  - Subroutines
    - Call, reuse similar techniques
- 

Feb 9, 2022

Sprenkle - CSCI111

13

13

## Making Decisions

- Sometimes, we do things only if some condition holds (i.e., “is true”)
- Examples
  - If the PB is new (has a safety seal)
    - Then, I will take off the safety seal
  - If it is raining and it is cold
    - Then, I will wear a raincoat
  - If it is Saturday or it is Sunday
    - Then, I will wake up at 9 a.m.
    - Otherwise, I wake up at 7 a.m.
  - If the shirt is purple or the shirt is on sale and blue
    - Then, I will buy the shirt

Feb 9, 2022

Sprenkle - CSCI111

14

14

# Conditionals

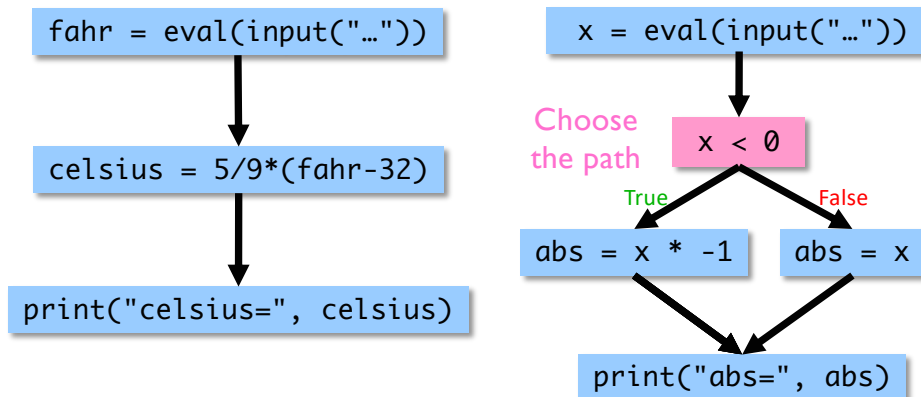
- Sometimes, we only want to execute a statement in certain cases
  - Example: Finding the absolute value of a number
    - $|4| = 4$
    - $|-10| = 10$
- To get the answer, we multiply the number by  $-1$  *only if it's a negative number*

➤ Code: 

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

# if Statements

- Change the *control flow* of the program



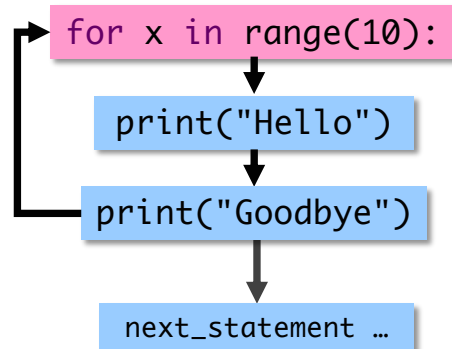


## Other Constructs That Change Control Flow

- **for** loops

- Repeats a loop body a fixed number of times before going to the next statement after the **for** loop

```
for x in range(10):  
    print("Hello")  
    print("Goodbye")  
next_statement ...
```



Feb 9, 2022

Sprenkle - CSCI111

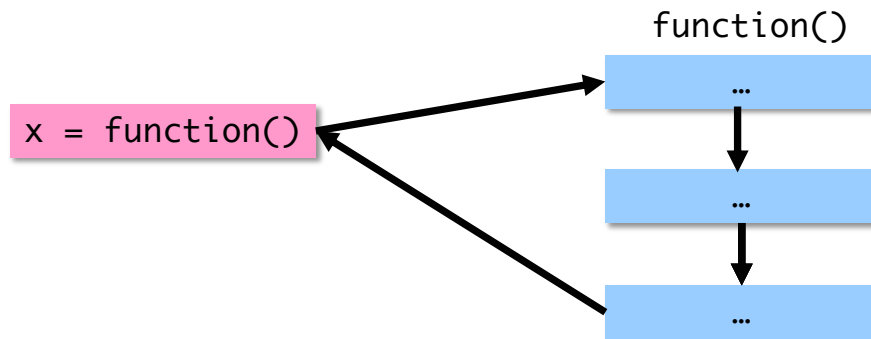
17

17

## Other Constructs That Change Control Flow

- Function calls

- “Go execute some other code and then come back with the result”



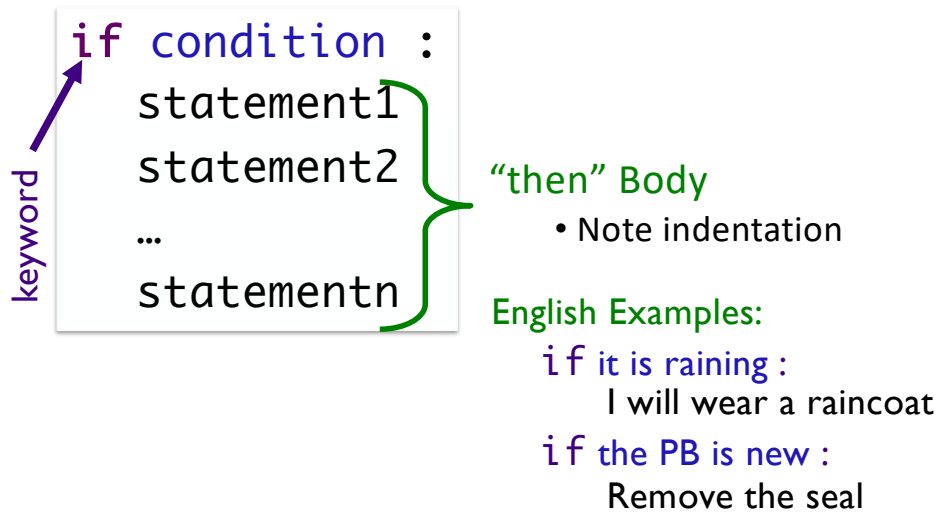
Feb 9, 2022

Sprenkle - CSCI111

18

18

## Syntax of **if** statement: Simple Decision



Feb 9, 2022

Sprenkle - CSCI111

19

19

## Conditions

- Syntax (typical, others later):
  - `<expr> <relational_operator> <expr>`
- Evaluates to either **True** or **False**
  - Boolean type

Feb 9, 2022

Sprenkle - CSCI111

20

20

# Relational Operators

- Syntax: <expr> <relational\_operator> <expr>
- Evaluates to either **True** or **False**
  - Boolean type

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

Low precedence  
After arithmetic operators

Feb 9, 2022

Sprenkle - CSCI111

Use Python interpreter

21

21

# Example: 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")
```

Feb 9, 2022

Sprenkle - CSCI111

evenorodd.py

22

22

## Common Mistake:

### Assignment Operator vs. Equality Operator

- Assignment operator: =
- Equality operator: ==

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

*Syntax error* (indicated by a red arrow pointing to the single equals sign in the if statement)

Feb 9, 2022

Sprenkle - CSCI111

23

23

## Syntax of **if** statement: Two-Way Decision

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

*keywords* (indicated by a purple arrow pointing to the **if** and **else** keywords)

*"then" Body* (indicated by a green bracket next to the first block of code)

*"else" Body* (indicated by a green bracket next to the second block of code)

**English Example:**

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

Feb 9, 2022

Sprenkle - CSCI111

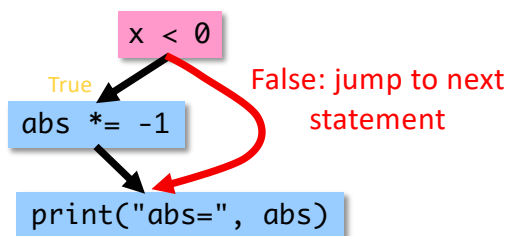
24

24

## If-Else statements (absolute values)

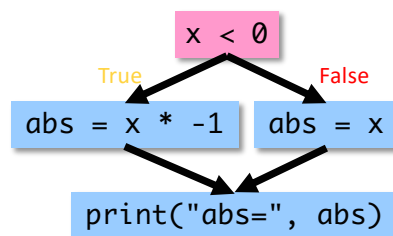
```
abs = x
if x < 0 :
    abs *= -1
print("abs=", abs)
```

If statement



```
if x < 0 :
    abs = x * -1
else:
    abs = x
print("abs=", abs)
```

If-else statement



Feb 9, 2022

Sprenkle - CSCI111

25

25

## Example: Using Conditionals

- Determine if a number is even or odd
- More efficient implementation
  - Don't need to check if remainder is 1 because if it's not 0, it *must* be 1

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

Feb 9, 2022

26

26

## Practice: Draw the Flow Chart

```
print("This program determines your birth year")
print("given your age and current year")
print()
age = eval(input("Enter your age: "))

if age > 120:
    print("Don't be ridiculous, you can't be that old.")
else:
    currentYear = eval(input("Enter the current year: "))
    birthyear = currentYear - age
    print()
    print("You were either born in", birthyear, end=' ')
    print("or", birthyear-1)
print("Thank you. Come again.")
```

What does this code do?

Feb 9, 2022

Sprenkle - CSCI111

27

27

## Function: max

- Given two numbers, returns the greater number

Feb 9, 2022

Sprenkle - CSCI111

28

28

# Flow of Control: Using **return**

Is this implementation of the function correct?

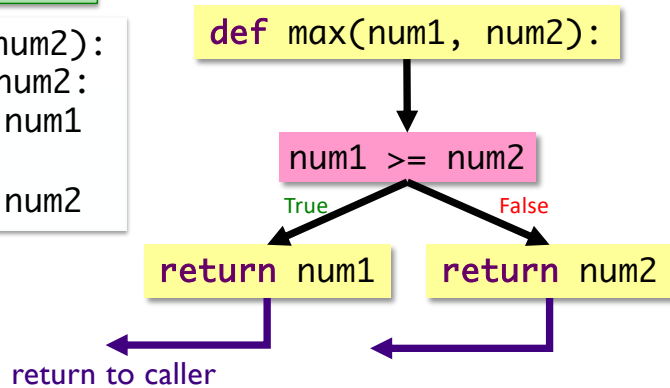
Review: What does a **return** statement do/mean?

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

# Flow of Control: Using **return**

Is this implementation of the function correct?

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



# Flow of Control: Using return

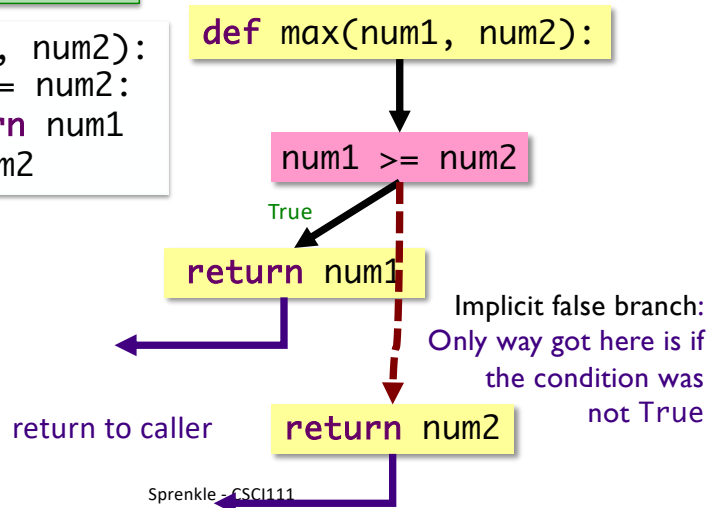
Is this implementation of the function correct?

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

# Flow of Control: Using return

Is this implementation of the function correct?

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





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

Feb 9, 2022

Sprenkle - CSCI111

[speedingticket.py](#)

33

33

## Exam Friday

- In-class, on paper
  - Emphasis on critical thinking
- Exam Preparation Document is on course web page
- Similar problems to class and lab
  - Review questions
  - Worksheets
  - Problems
- Content: up through Lab 4
- No broader issue this week

Feb 9, 2022

Sprenkle - CSCI111

34

34

## Looking Ahead

- Lab 4
  - Practicing *functions*
  - Due Friday
- Exam Friday
- No broader issue this week