CS134: Functions, Booleans & Conditionals

Announcements & Logistics

- **Homework 2** is due Monday 11 pm
  - Ten multiple-choice questions on Glow
  - Try to answer them using pencil and paper first
  - Can verify answers using interactive Python if you wish
- **Lab 2** has been posted, due Wed 11pm / Thur 11pm
  - Plan to spend 30-60 min on it before arriving at lab
- Please double check your Peoplesoft enrollment and make sure you're in the correct lab and lecture section (today is the last day to make changes!)
- Windows users who had trouble with computer configuration: see email from Steve and let us know if you need help

Do You Have Any Questions?

Last Time

- Discussed **functions** in greater detail
- Reviewed the built-in functions:
  - `input()`, `print()`, `int()`, `float()`, `str()`
- Saw that some functions return an explicit value (called **fruitful**)
  - `int()`, `input()`, our definition of `square()`
- Other functions "do something" but don't explicitly return
  - `print()`, user-defined functions *without* explicit return statement
  - Such functions "secretly" return a **None** value (more on this today!)

Jupyter Notebook:
Let’s See Some Examples
(from last lecture’s notebook)

Today’s Plan

- Write a non-trivial function together in Atom
- Review two ways to test functions:
    - **interactively** (Python prompt >>> in Terminal)
    - by running it as a **script** (Save file in Atom, run in Terminal)
- Wrap up discussion of functions
    - Discuss return statements and variable scope in more detail
- Start learning about conditionals (Lab 2!)
    - Boolean data type
    - Making decisions in Python using `if else` statements

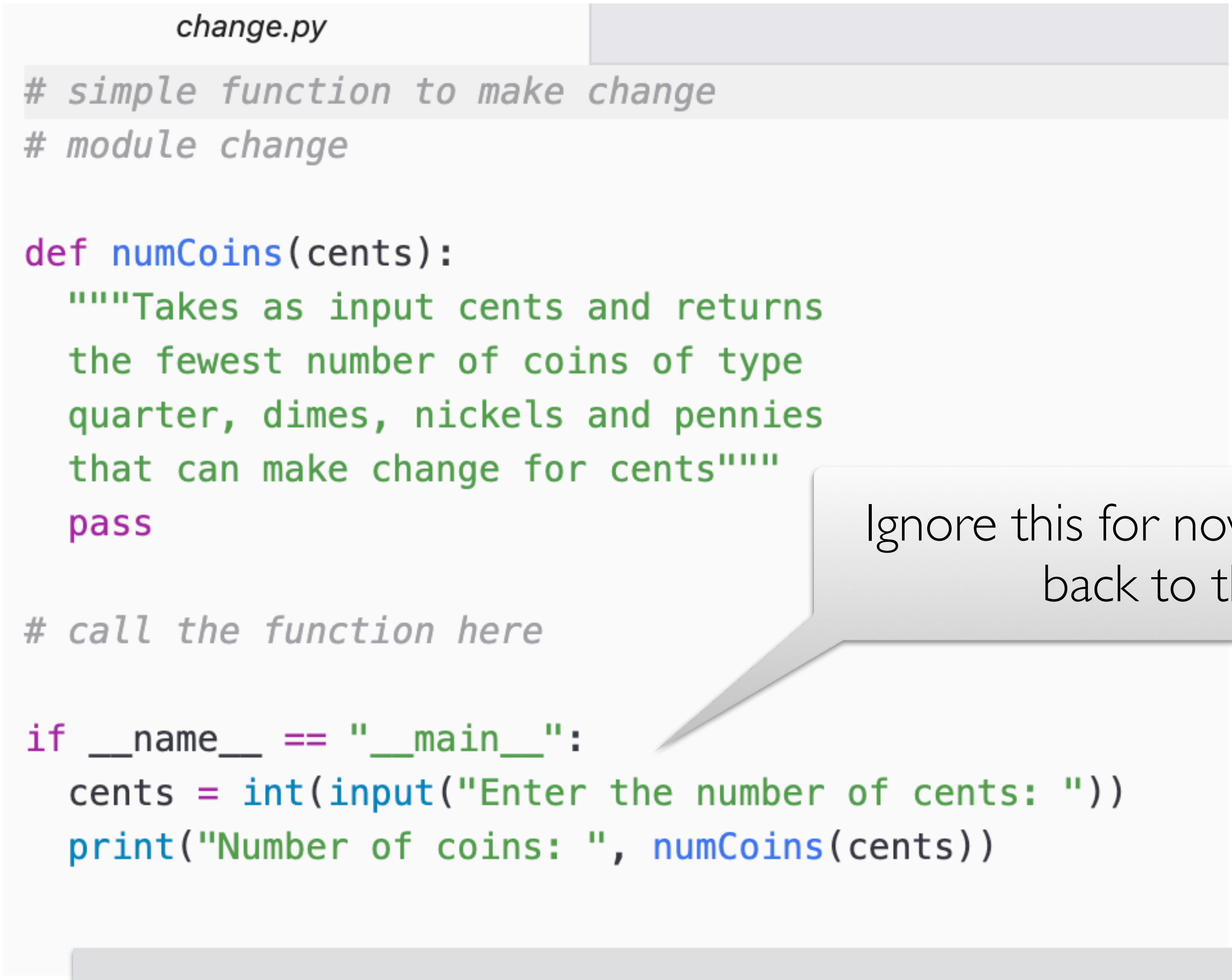
Exercise: Making Change

- Suppose you are a cashier and you need to make change for a given number of cents using only quarters, dimes, nickels, and pennies
- Most cashiers use the following greedy strategy to make change using the fewest number of coins:
  - Use as many quarters as possible first, then as many dimes as possible next, and so on, using pennies last
  - Assume you have an unlimited supply of each coin

Exercise: Making Change

- **Problem**. Let us write a function `makeChange(cents)` that takes as a parameter an integer `cents` and returns the fewest number of coins needed to make change for `cents` cents
- **Approach**: Decompose the problem into smaller pieces
  - What is the maximum number of quarters we can use?
    - `q = cents // 25`
  - How much money is left after we use `q` quarters?
    - `cents = cents % 25`
  - For the remaining cents, what is the maximum number of dimes can we use?

Example Code



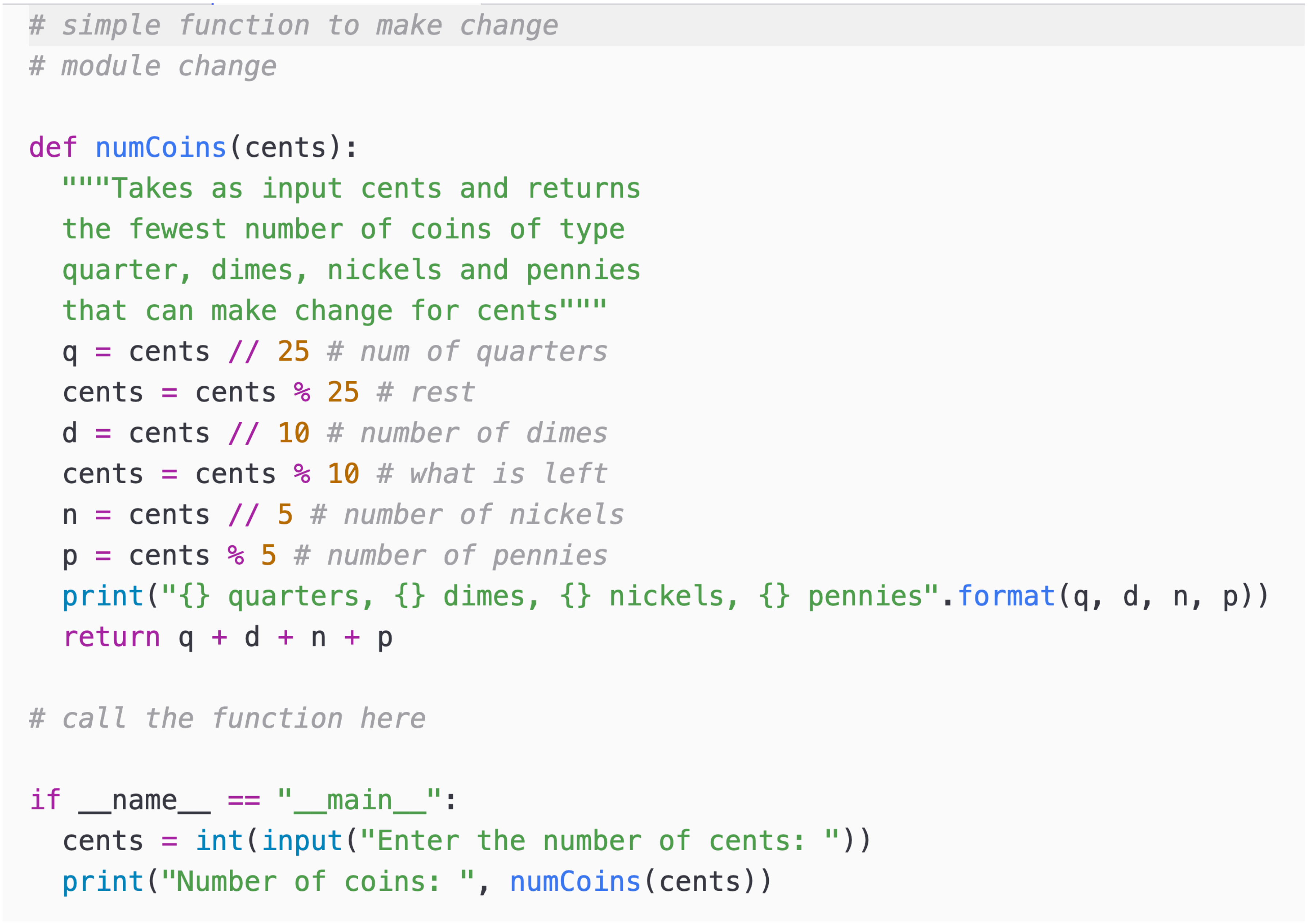
change.py

```python
# simple function to make change
# module change

def numCoins(cents):
  """Takes as input cents and returns
  the fewest number of coins of type
  quarter, dimes, nickels and pennies
  that can make change for cents"""
  pass

# call the function here

if __name__ == "__main__":
  cents = int(input("Enter the number of cents: "))
  print("Number of coins: ", numCoins(cents))
```

Ignore this for now... We will come back to this soon.

Let's implement this together!

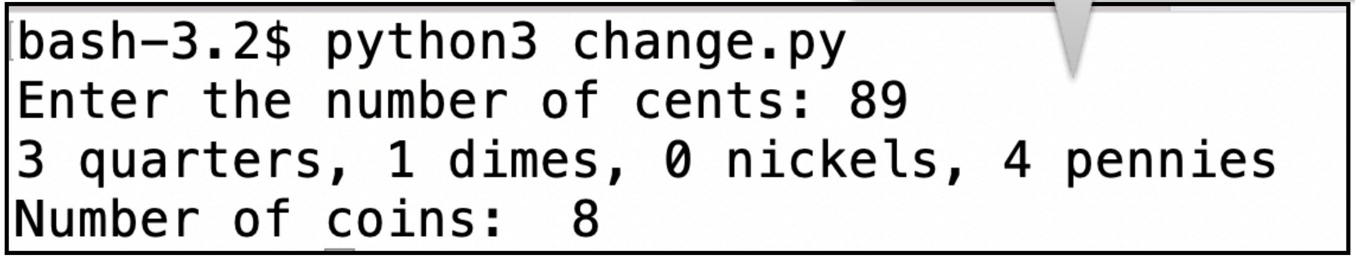
Solution



```python
# simple function to make change
# module change

def numCoins(cents):
  """Takes as input cents and returns
  the fewest number of coins of type
  quarter, dimes, nickels and pennies
  that can make change for cents"""
  q = cents // 25 # num of quarters
  cents = cents % 25 # rest
  d = cents // 10 # number of dimes
  cents = cents % 10 # what is left
  n = cents // 5 # number of nickels
  p = cents % 5 # number of pennies
  print("{} quarters, {} dimes, {} nickels, {} pennies".format(q, d, n, p))
  return q + d + n + p

# call the function here

if __name__ == "__main__":
  cents = int(input("Enter the number of cents: "))
  print("Number of coins: ", numCoins(cents))
```

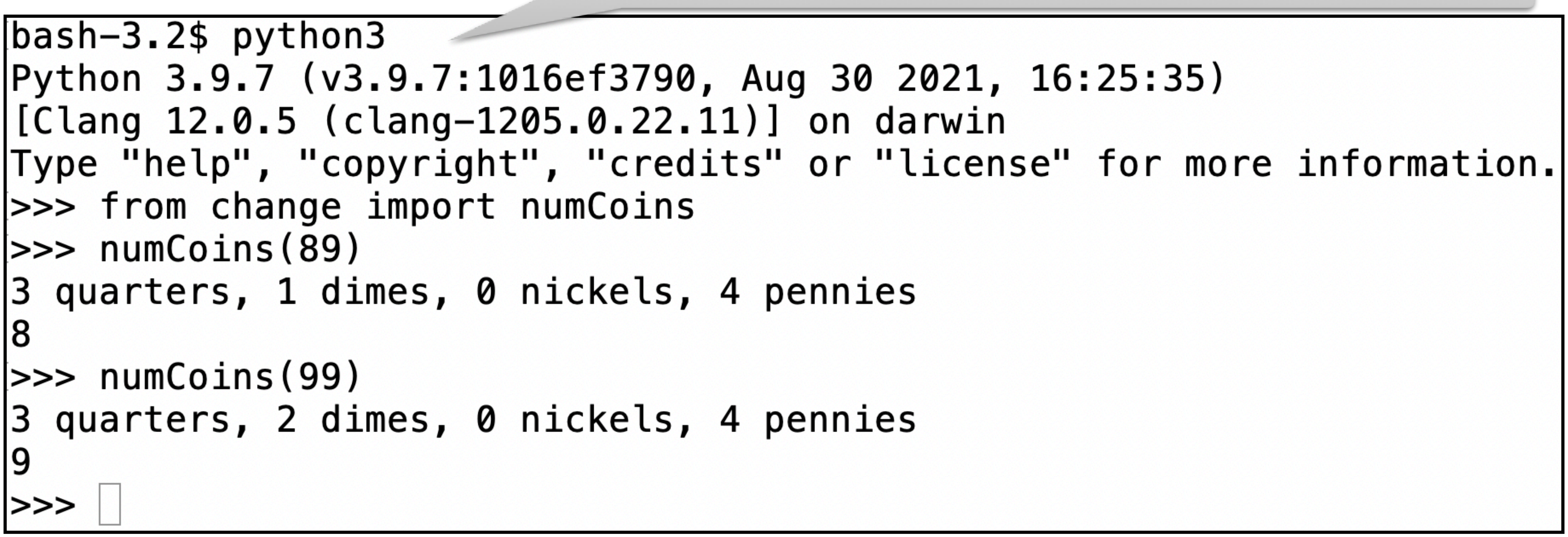
Two Ways To Test Our Code

1) Write code in a file change.py. Execute the program from the Terminal using python3.



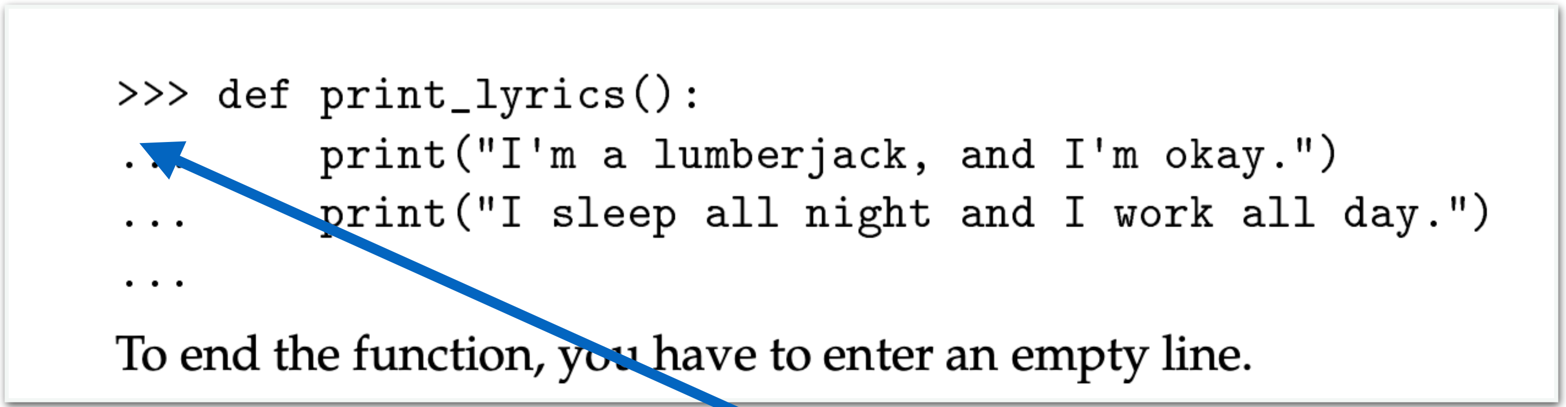
```
bash-3.2$ python3 change.py
Enter the number of cents: 89
3 quarters, 1 dimes, 0 nickels, 4 pennies
Number of coins:  8
```

2) Test interactively by importing the function in ***interactive Python***. We'll see this again in Lab 2.



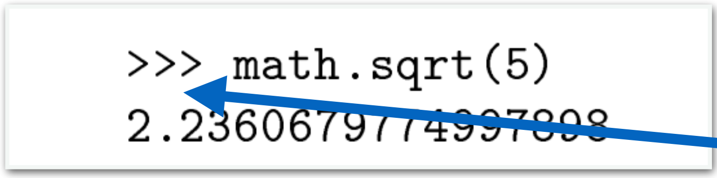
```
bash-3.2$ python3
Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:25:35)
[Clang 12.0.5 (clang-1205.0.22.11)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from change import numCoins
>>> numCoins(89)
3 quarters, 1 dimes, 0 nickels, 4 pennies
8
>>> numCoins(99)
3 quarters, 2 dimes, 0 nickels, 4 pennies
9
>>> 
```

Aside: Running Code in Textbook



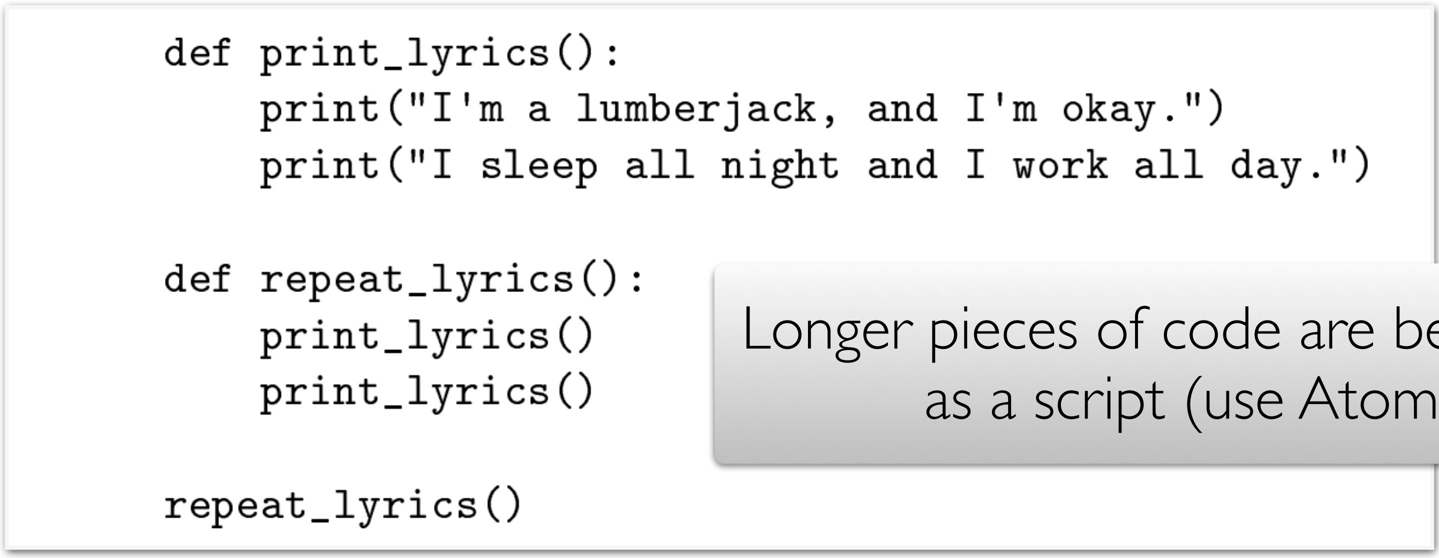
```
>>> def print_lyrics():
...     print("I'm a lumberjack, and I'm okay.")
...     print("I sleep all night and I work all day.")
...
```

To end the function, you have to enter an empty line.



```
>>> math.sqrt(5)
2.2360679774997898
```

Three carrot signs (>>>) represent interactive Python mode (in Terminal)



```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print("I sleep all night and I work all day.")

def repeat_lyrics():
    print_lyrics()
    print_lyrics()

repeat_lyrics()
```

Longer pieces of code are better run as a script (use Atom)

Variable Scope

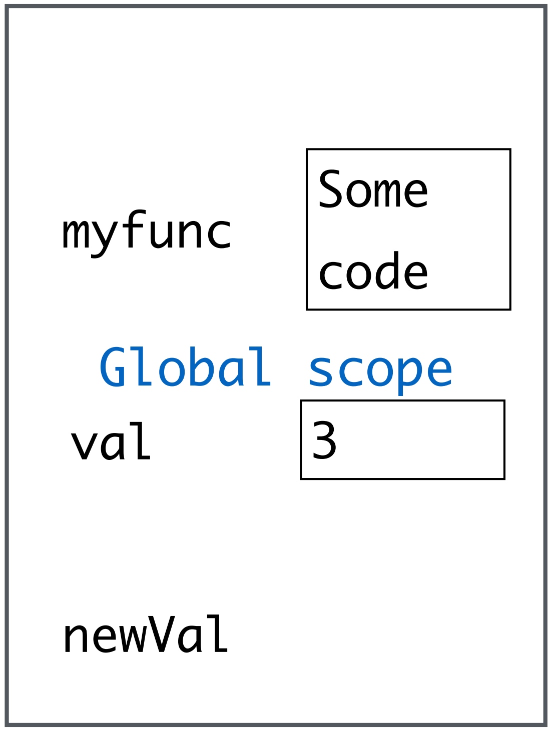
- **Local variables:** An assignment to a variable *within a function* definition creates/modifies a *local variable*
- Local variables exist and are valid *only* within a function's body, and cannot be referred to outside of it
- **Parameters** are also local variables that are assigned a value when the function is invoked

```
def square(num):

   return num*num
```

```
>>> square (5)

25

>>> num

NameError: name 'num' is not defined
```

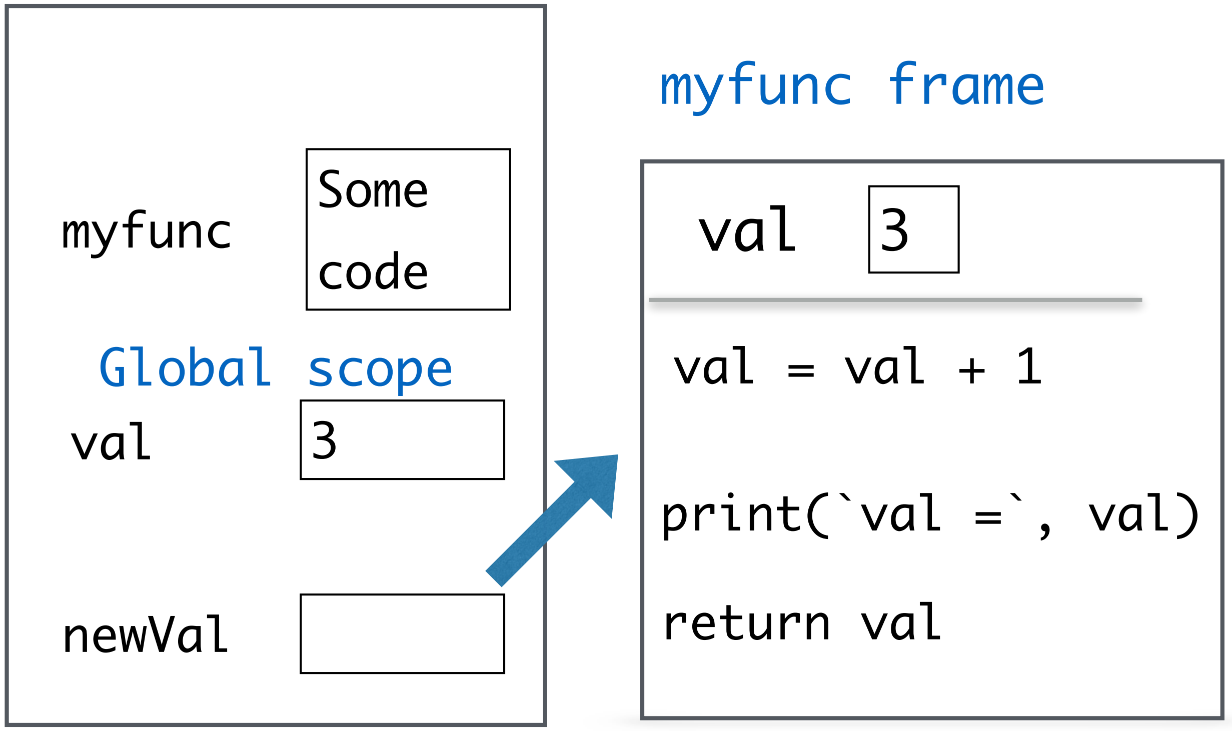
Variable Scope: A Tricky Example

```
def myfunc (val):
    val = val + 1
    print('val = ', val)
    return val


val = 3
newVal = myfunc(val)
```



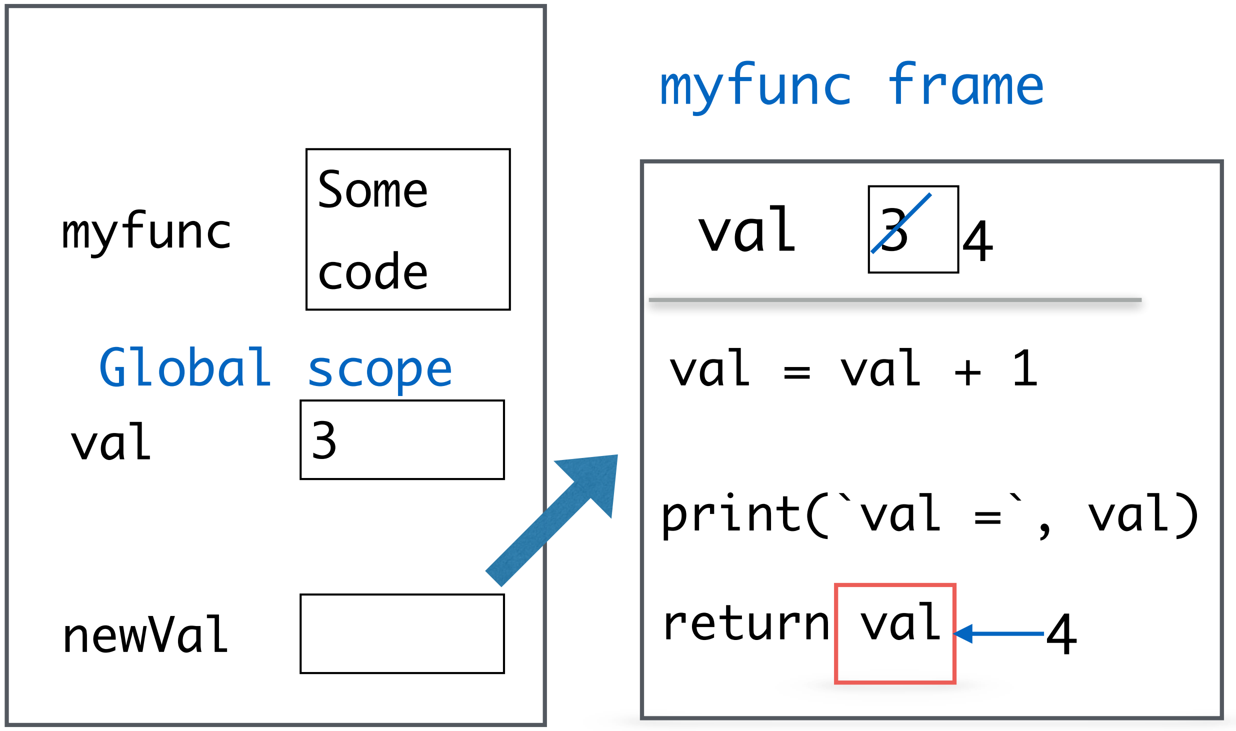
Variable Scope: A Tricky Example

```
def myfunc (val):
    val = val + 1
    print('val = ', val)
    return val


val = 3
newVal = myfunc(val)
```



Variable Scope: A Tricky Example

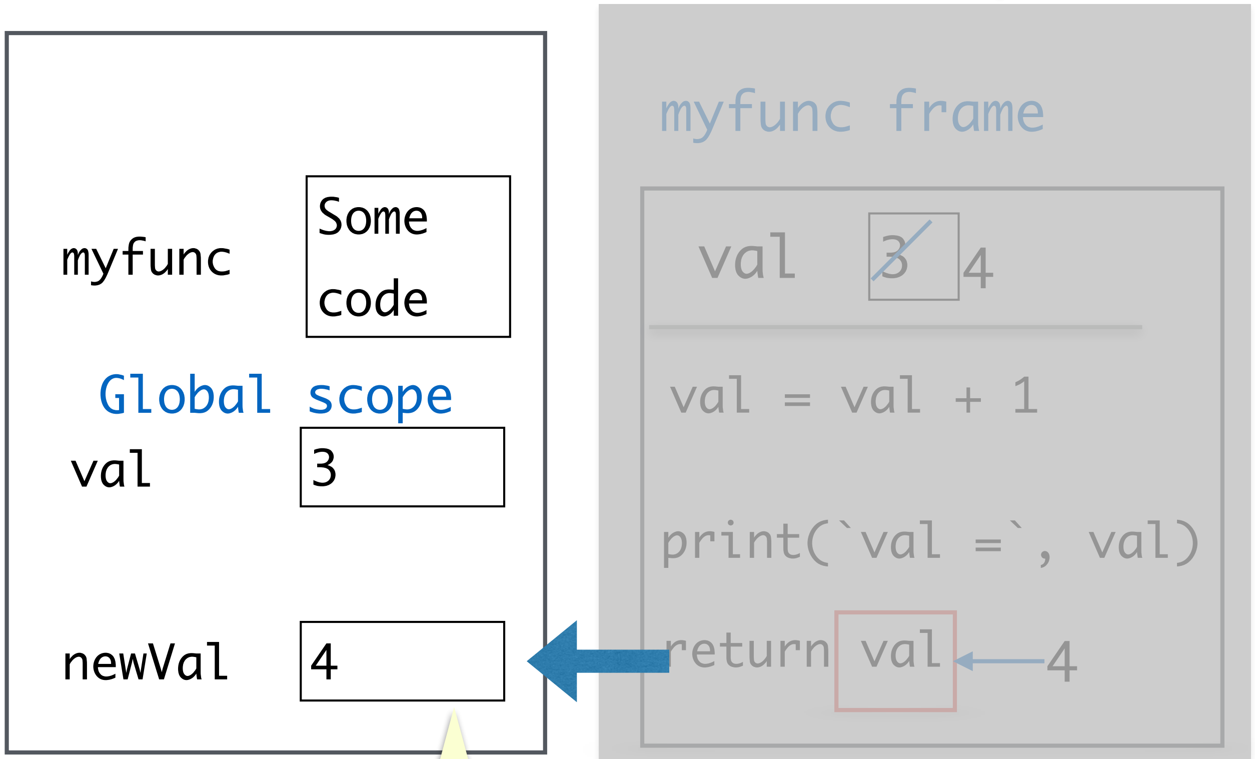
```
def myfunc (val):
    val = val + 1
    print('val = ', val)
    return val


val = 3
newVal = myfunc(val)
```



Variable Scope: A Tricky Example

```
def myfunc (val):
    val = val + 1
    print('val = ', val)
    return val


val = 3
newVal = myfunc(val)
```

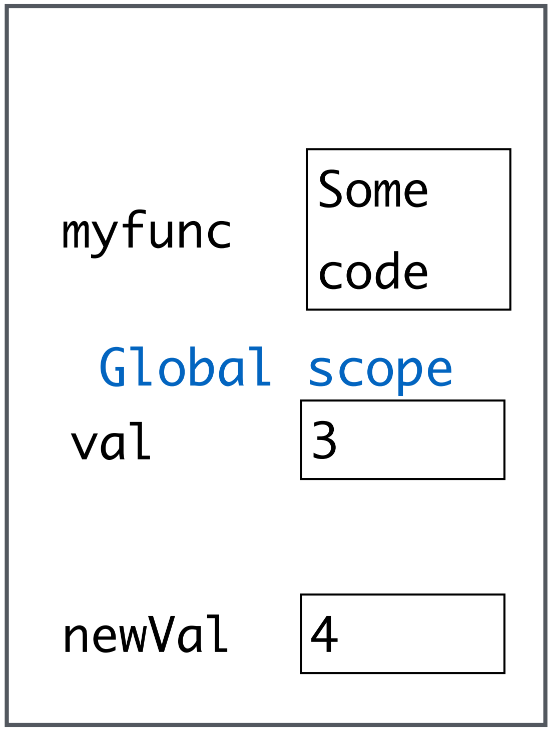
Function frame destroyed (and all local variables lost) after return from call



Information flow out of a function is only through return statements!

Variable Scope: A Tricky Example

```
def myfunc (val):
    val = val + 1
    print('val = ', val)
    return val


val = 3
newVal = myfunc(val)
```



Return Statements

- `return` only has meaning inside of a function definition
- A function definition may have multiple returns, **but only the first one encountered is executed!**
    - We will see functions with multiple returns very soon!
- Any code that exists after a return statement **is unreachable** and will not be executed
- The value returned by the function's return statement replaces the function call in a computation
- Functions without an explicit return statement implicitly return **None**

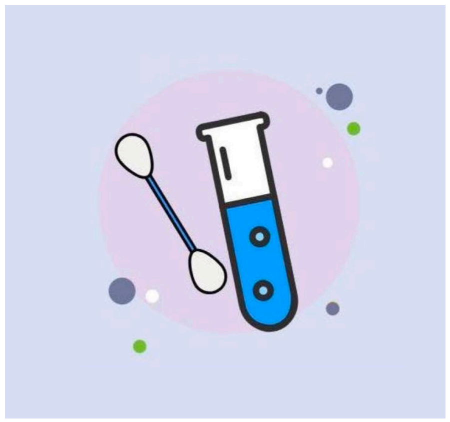
Moving on:
Making Decisions

Making Decisions

If it is raining, then bring an umbrella.

If the light is yellow, slow down. If it is red, stop.

If you are inside an academic building, wear a mask.



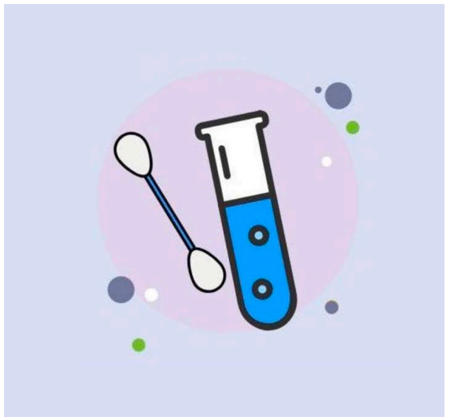
If your name starts with letters A-L, test on Tuesdays.

Making Decisions

If it is raining, then bring an umbrella.

If the light is yellow, slow down. If it is red, stop.

If you are inside an academic building, wear a mask.



If your name starts with letters A-L, test on Tuesdays.

Decisions Based on Yes/No Questions

If it is raining, then bring an umbrella.

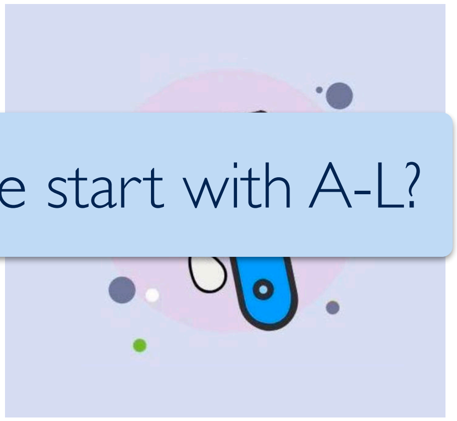
Is it raining?

If the light is yellow, slow down. If it is red, stop.

Is it yellow? red? green?

If you are inside an academic building, wear a mask.

Are you inside ?



Does your name start with A-L?

If your name starts with letters A-L, test on Tuesdays.

Boolean Types

- Python has two values of **bool** type, written **`True`** and **`False`**
- These are called logical values or Boolean values, named after 19th century mathematician George Boole
- **`True`** and **`False`** must be capitalized!
  - Internally True = 1, False = 0
- Boolean values naturally result when answering a yes or no question
  - Is 10 greater than 5?   Yes/True
  - Is 23 an even number? No/False
  - Does 'Williams' begin with a vowel? No/False
- Boolean values result naturally when using **relational** and **logical** operators

Relational Operators

< (less than), > (greater than)

<= (less than or equal to), > = (greater than or equal to)

== (equal to), ! = (not equal to)

Reminder that the single = is an assignment, double == is equality

```
>>> 3 > 5
False
>>> 5 != 6
True
>>> 5 == 5
True
```

Relational Operators

< (less than), > (greater than)

<= (less than or equal to), > = (greater than or equal to)

== (equal to), ! = (not equal to)

Reminder that the single = is an assignment, double == is equality

```
>>> 0 == True
False
>>> True == True
True
>>> int(False)
0
>>> int(True)
1
```

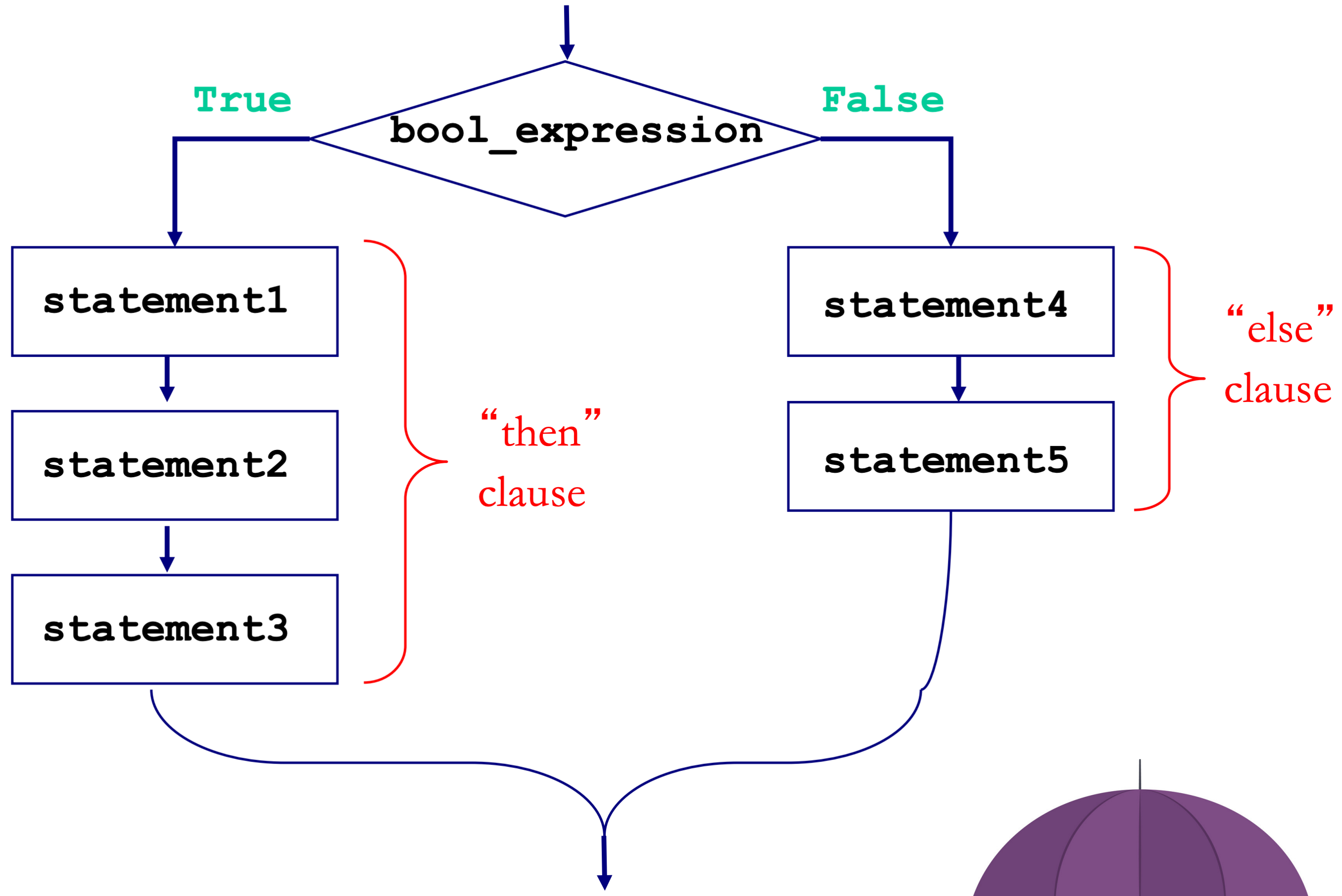
Boolean Expressions and If Statement

- Python expressions that result in a `True/False` output are called **boolean expressions**
- For example, checking if a user's entered number, `num`, is even
- How do we do this? (What is a property of even numbers that we can use to test this condition?)
    - Even numbers are divisible by 2 (give remainder zero)
    - Thus, `num` % 2 should be zero if and only if `num` is even
- Now we have a Boolean expression we can test for: `num % 2 == 0`
- We can implement "conditional statements" in Python using Boolean expressions and an **if-else statement**

Python Conditionals (`if` Statements)

```
if <boolean expression>:
        statement1
        statement2
        statement3
else:
        statement4
        statement5
```

Note: (syntax) Indentation and colon after if and else



If it is raining, then bring an umbrella.
Else, bring your sunglasses.

Conditional Statements: If Else

- Consider the following functions that check if a number is even or odd

```python
def printEven(num):
    """Takes a number as input, prints Even if
    it is even, else prints Odd"""
    if num % 2 == 0: # if even
        print("Even")
    else:
        print("Odd")
```

```python
def isEven(num):
    """Takes a number as input, returns True if
    it is even, else returns False"""
    return num % 2 == 0
```