Lecture 8: Introduction to Computer Programming Course - CS1010

DEPARTMENT OF COMPUTER SCIENCE | 09/27/2019

Rensselaer

Announcements

1

Homework 3 is Posted and is due next week

2

Exam 1 will be on October 1 (in-class)

3

It will be a closed book, closed computers exam.

4

You can bring 1 page (A4) of handwritten notes!

Goals

- More about Decision Logic:
  - Visualize the execution of If Statements: Flow Charts
  - More about Boolean Logic
  - Practice Problems
- Exam Review
- In-Class Exercise

Decision Making

Similar to real life in programming also, we need to make decisions

Based on these decision we execute the next block/chunk of code.

In programming languages like Python, decision making statements decide the direction in which your program flows

Decision making statements available in python are:

if statement

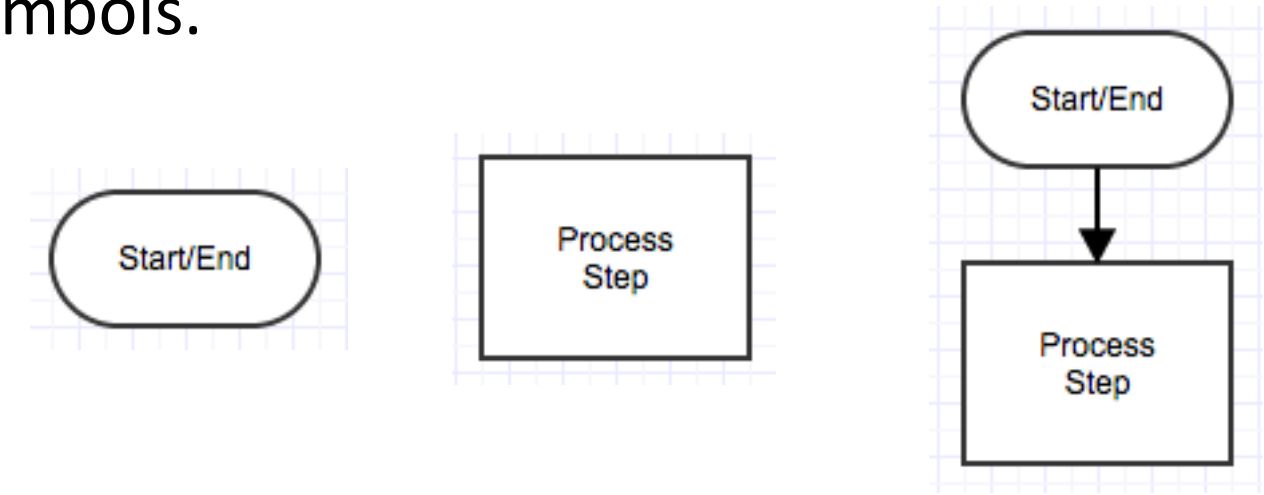
if..else statements

nested if statements

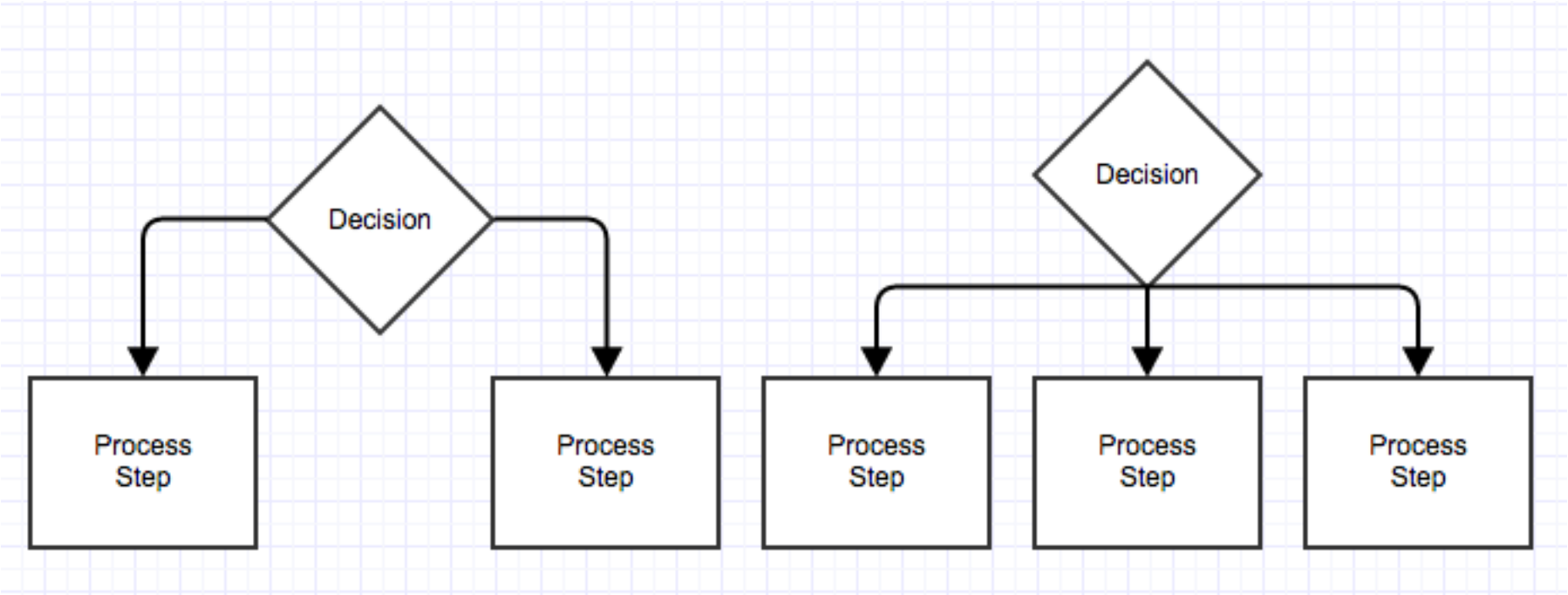
if-elif ladder

How to flowchart

- A diagram that shows step-by-step progression through a procedure or system especially using connecting lines and a set of conventional symbols.

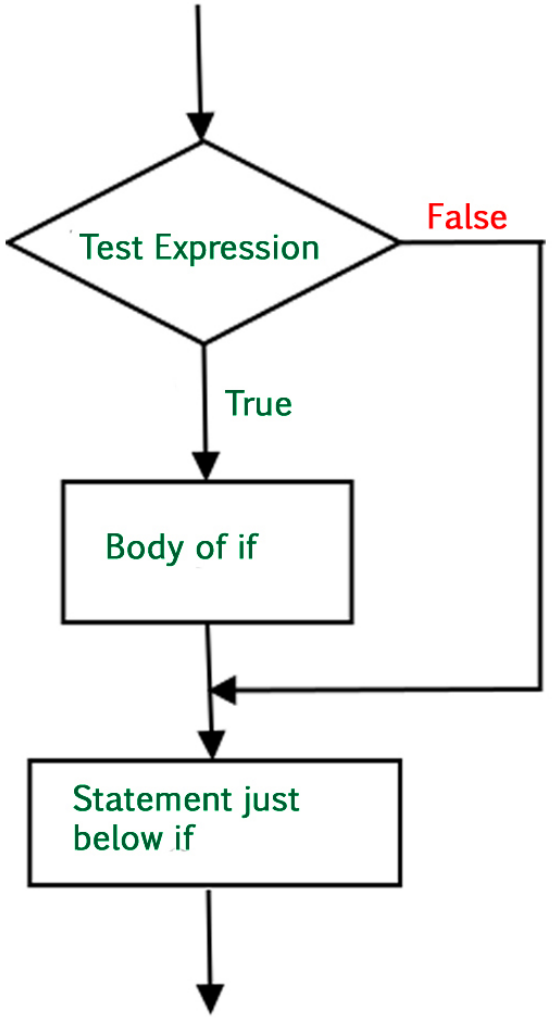


Decision-Flow Chart



If Statements

- if statement is used to decide whether a certain block of code will be executed or not based on a given criteria.
- The control will go inside the body of the code ’when’ the condition is True



If Statement

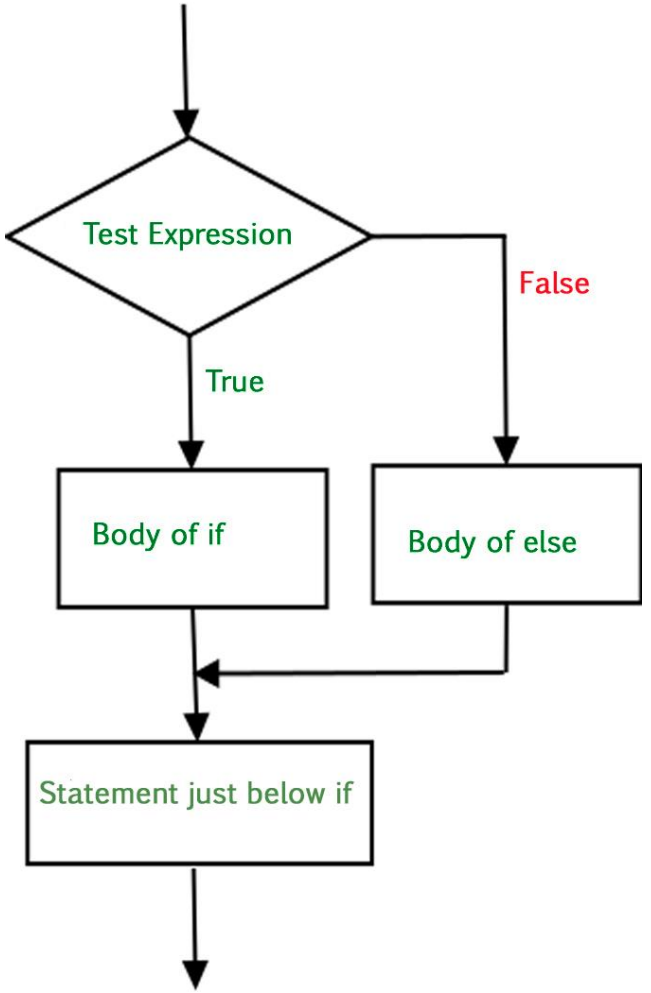
```
i =10
If (i>20):
    print (‘I am statement inside if’)
print(‘I am outside the if block’)
```

OUTPUT:

I am outside the if block

If-Else Statements

- We can use the else statement to execute another block of code when the if condition is false.



If-Else Statement

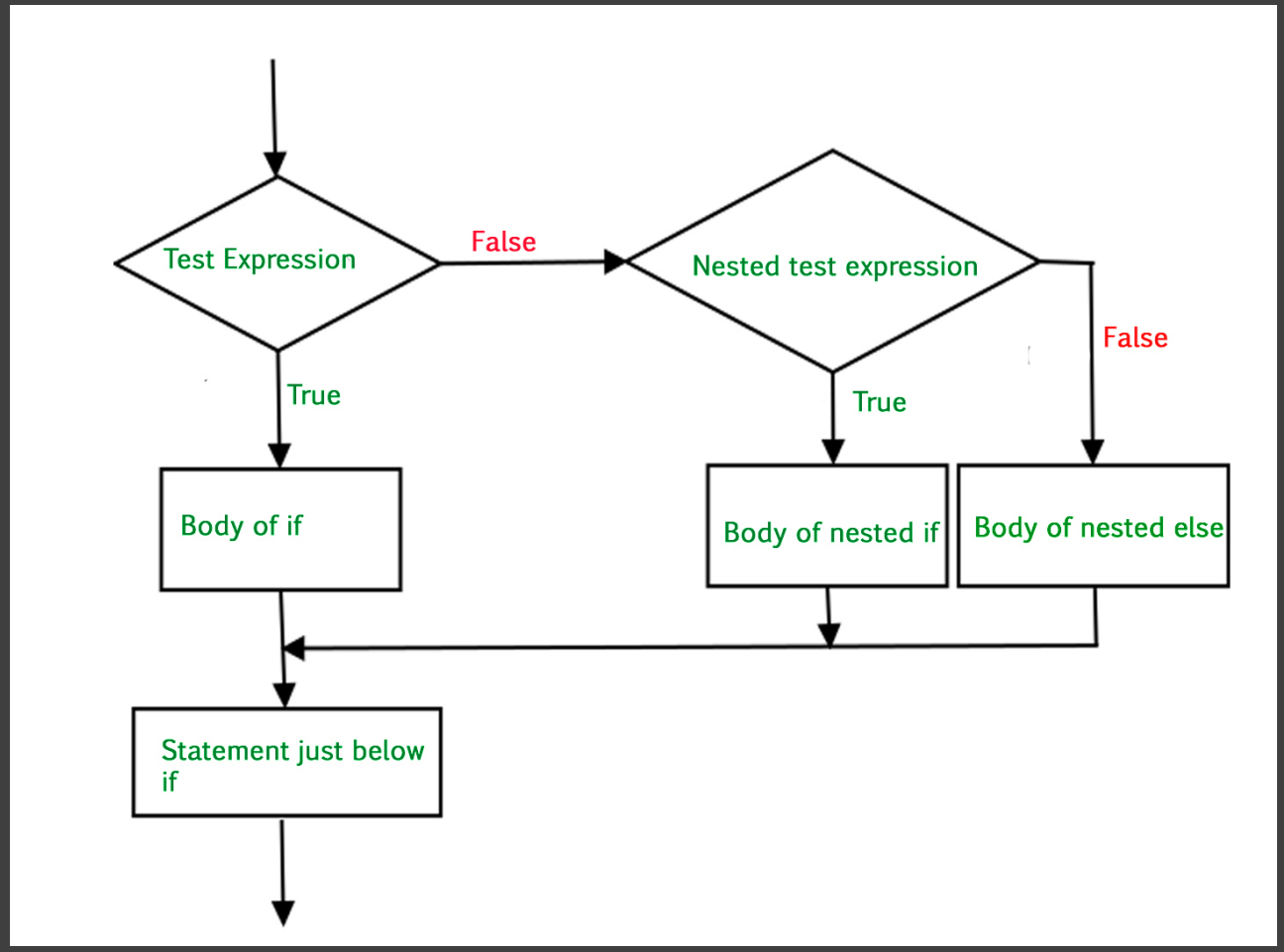
```
j = 30;
if (j < 15):
    print (”j is in if block")
else:
    print (”j is in else block")
print (”j is neither in if and nor in else Block")
```

OUTPUT:

```
j is in else block
j is neither in if and nor in else Block
```

Nested If Statements

Multiple if, if-else statements can be enclosed within an if or an else condition



Nested If Statement

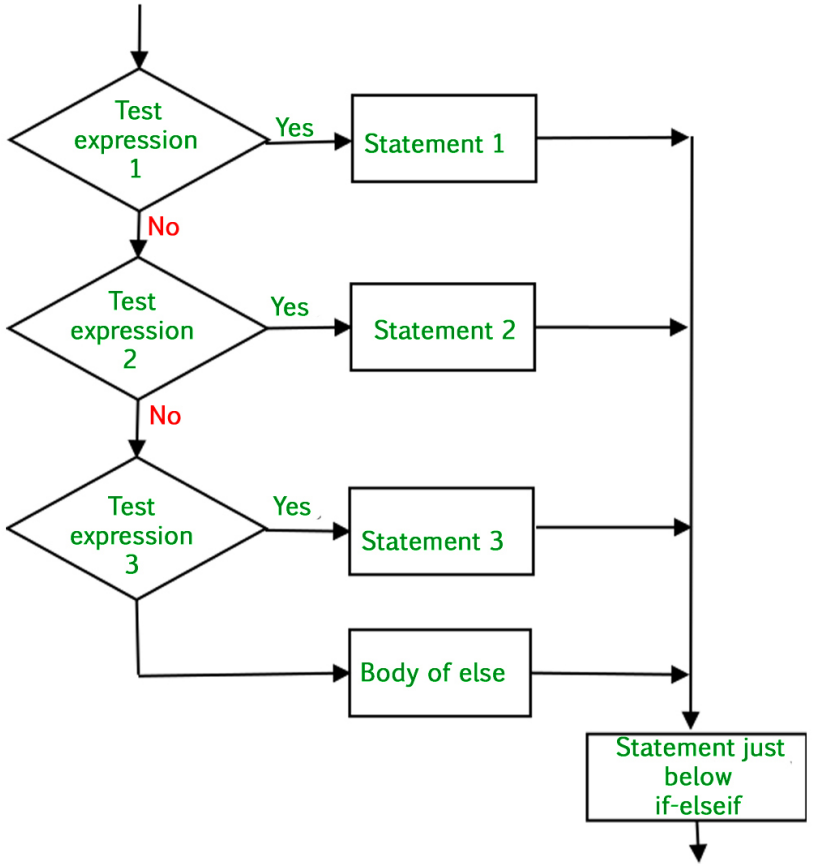
```
k = 10
if (k == 10):
    #  First if statement
    if (k < 15):
        print ("k is in first if statement")
    # Nested - if statement
    # Will only be executed if statement above
    # it is true
          if (k < 12):
                print ("k is in the nested if")
      else:
        print ("k is in else block of nested if")
```

OUTPUT:

```
k  is in first if statement
k is in the nested if
```

If-Elif-Else

- If Statements are executed from the top
- Under the If-elif-else ladder the control checks for each block i.e. starting from if and moving on to each ‘elif’.
- If none of the above is true then it executed the else.



If-elif-else Statement

```
l = 20
if (l == 10):
    print ("l is 10")
elif (l == 15):
    print ("l is 15")
elif (l == 20):
    print ("l is 20")
else:
    print ("l is not present")
```

OUTPUT:

```
l is 20
```

Boolean Algebra

Go to Lecture 3 Slides and look for	
Boolean Algebra and also	
Truth Tables	NOT
	AND
	OR

Boolean Values and Expressions

- In Python, the two Boolean values are True and False (the capitalization must be exactly as shown), and the Python type is **bool**.
- type(**True**)
- <class 'bool’>
- A **Boolean expression** is an expression that evaluates to produce a result which is a Boolean value. For example, the operator == tests if two values are equal. It produces a Boolean value:
- 5 == (3 + 2) *# Is five equal 5 to the result of 3 + 2?*
- True

Booleans (From Lecture 3)

- Boolean represents logical values (TRUE or FALSE)
- The **bool()** method is used to return or convert a value to a Boolean value
- The bool() method in general takes only one parameter, on which the standard truth testing procedure can be applied.
- **If no parameter is passed, then by default it returns False**.

Basic Boolean Algebra

- **Boolean Algebra** is a branch of algebra that involves bools, or true and false values.
- They're typically denoted as ***T or 1 for true*** and ***F or 0 for false***.
- Using this simple system we can boil down complex statements into easier/understandable logical statements.

Truth Table

- A **truth table** is a way of organizing information to list out all possible scenarios.
- p denotes proposition (condition) then ~p (read as not p) means everything opposite of the proposition.

p	~p
T	F
F	T

Binary Operators

- AND Operator
  - *Requires* both p and q to be True for the result to be True. All other cases result in False.

p	q	P AND q
T	T	T
T	F	F
F	T	F
F	F	F

- Keyword in Python: **and**

- OR Operator
  - *Requires* only one proposition to be True for the result to be True.

p	q	P OR q
T	T	T
T	F	T
F	T	T
F	F	F

- Keyword in Python: **or**

Operators and Expressions (In Python)

Operators	Expressions	Example
==	If the two operands are equal then the condition will be true	x=3, y=5; (x==y) is not true.
!=	If the two operands are not equal then the condition is true	(x!=y) is true
>	If the value on the left is greater than that on the right, then the condition is true	(x>y) is not true.
<	If the value on the right is greater than that on the left, then the condition is true	(x<y) is true
>=	If the value on the left is greater than or equal to the one on the right, then the condition is true	(x>=y) is false
<=	If the value on the right is greater than or equal to the one on the left, then the condition is true	(x<=y) is true

Simplifying Boolean Expressions

- Although these operations are probably familiar, the Python symbols are different from the mathematical symbols.
- A common error is to use a single equal sign (=) instead of a double equal sign (==).
- Remember that = is an assignment operator and == is a comparison operator.

Boolean Algebra (AND Operator)

- Let x and y be some Boolean and/or integer then
  - x and False == False
  - False and x == False
  - y and x == x and y
  - x and True == x
  - True and x == x
  - x and x == x
- What happens when x is False? Does everything above still hold?

Boolean Algebra (OR and NOT Operator)

- x or False == x
- False or x == x
- y or x == x or y
- x or True == True
- True or x == True
- x or x == x

- NOT Operator:
  - not (not x) == x

The return Statement

- The return statement, depending on whether the function gives a value or is void, allows us to terminate the execution of a function before (or when) we reach the end.
- One reason to use an *early return* is if we detect an error condition:

```
def sqrt(y):
    if y <= 0:
        print(”No negatives or zeroes, please.")
        return
    result = y**0.5
    print("The square root of", y, "is", result)
```

- Using return here ends the function so that the lines after return will not be executed.

Logical Opposites

- Each of the six relational operators has a logical opposite:
  - An example: Suppose you can vote at age 18 or above therefore you can NOT vote at any age other than ’18 or above’.

Logical Operator	Opposite
==	!=
!=	==
<	>=
>	<=
>=	<

Eliminating NOT

```
if not (age >= 18):
    print("Hey, you're too young to vote")
```

- **if** age < 18:
- print("Hey, you're too young to vote!")

De-Morgan's Law

- Simplifying Expressions:
  - not (x and y) == (not x) or (not y)
  - not (x or y) == (not x) and (not y)
  - Example:
    - (not(x < 15 and y >= 3) has the same value as (x >= 15 or y < 3)

Problem 1

- Give the logical opposites of these conditions
- a > b
- a >= b
- a >= 18 and y == 3
- a >= 18 and y != 3
- a<15 or y>12

Problem 2

- What do these expressions evaluate to?
- 3 == 3
- 3 != 3

- 3 >= 4

- not (3 < 4)

Problem 3

- Write a function **isright** which, given the length of three sides of a triangle, will determine whether the triangle is right-angled. Assume that the third argument to the function is always the longest side. It will return True if the triangle is right-angled, or False otherwise.

Next Week's Lecture

- Tuples
- Modules
- Images

References

- https://www.gliffy.com/blog/how-to-flowchart-basic-symbols-part-1-of-3