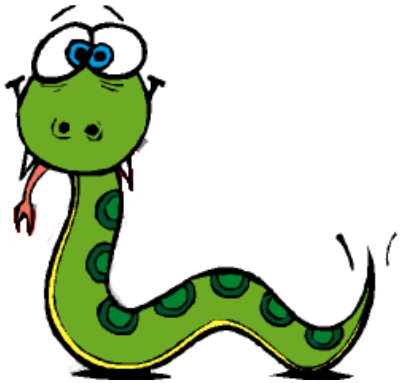
IDCE 302: Chapter 4

Conditionals & Recursion

Jie Tian, PhD

Clark University



Outline

- Modulus operator (%)
- Boolean expressions & Logical operators
- Conditional execution
  - Chained conditionals
  - Nested conditionals
- Recursion & Infinite recursion
- Keyboard input

The Modulus Operator (%)

```
>>> quotient = 7 / 3
>>> print quotient
2

>>> remainder = 7 % 3
>>> print remainder
1
```

Applications

- Check if x is divisible by y (x % y):

```
x % y == 0
```

- Test if x is an odd or even number

```
x % 2 == 0
```

```
1%2 ->1,  3%2 ->1,  115%2 -> 1
2%2 ->0,  4%2 ->0,  116%2 -> 0
```

- Get the right-most digit of a integer of base 10

```
y = x % 10
```

```
23%10 -> 3
60%10 -> 0
```

Boolean Expressions

- `True` and `False` are **Boolean** values in Python
- An Boolean expression yields either `True` or `False`.
- **Relational Operators** / Comparison Operators are often used in Boolean expressions:

```
==      equal to
!=      not equal to
>       greater than
>=      greater than or equal to
<       less than
<=      less than or equal to
```

```
x == y    # x is equal to y?
x != y    # x is not equal to Y?
x > y     # x is greater than Y?
x < y     # x is less than y?
x >= y    # x is greater than or equal to y?
x <= y    # x is less than or equal to y?
```

Note 1: **=** is an assignment operator and
== is a relational operator.

Logical Operators

- **Logical operators** are often used in building Boolean expressions.
- Three logical operators: **and**, **or**, and **not**

Expression	Result
True and False	False
True or False	True
not False	True
not True	False

Examples:

```
>>> x = True
>>> y = False

>>> z = x and y
>>> z
False

>>> z = x or y
>>> z
True
```

```
>>> x= 155

>>> suitability = 0.9

>>> x = area>150 and suitability > 0.8
>>> x
True

>>>y = not x
>>>y
False


>>> x = 0
>>> y = not x
>>> y
True
```

bool()

Conditional Execution (if)

- We often need the ability to **check conditions** and **change the behavior** of the program **accordingly.**

Run script in PythonWin

```
x = 5
if x>0:
   print x, "is positive"
y = -5
if y < 0:
   print y, 'is negative'
```

Output in the Interactive Window

```
5 is positive
-5 is negative
```

Alternative Execution (`if/else`)

The <u>if/else structure</u> for alternative execution:

```
if x%2 == 0:
    print x, "is even."
else:
    print x, "is odd."
```

```
>>> def printParity(x):
...   if x%2 == 0 :
...     print x, "is an even number. "
...   else:
...     print x, "is an odd number."
...
>>> printParity(3)
3 is odd number.
>>> printParity(4)
4 is an even number.
```

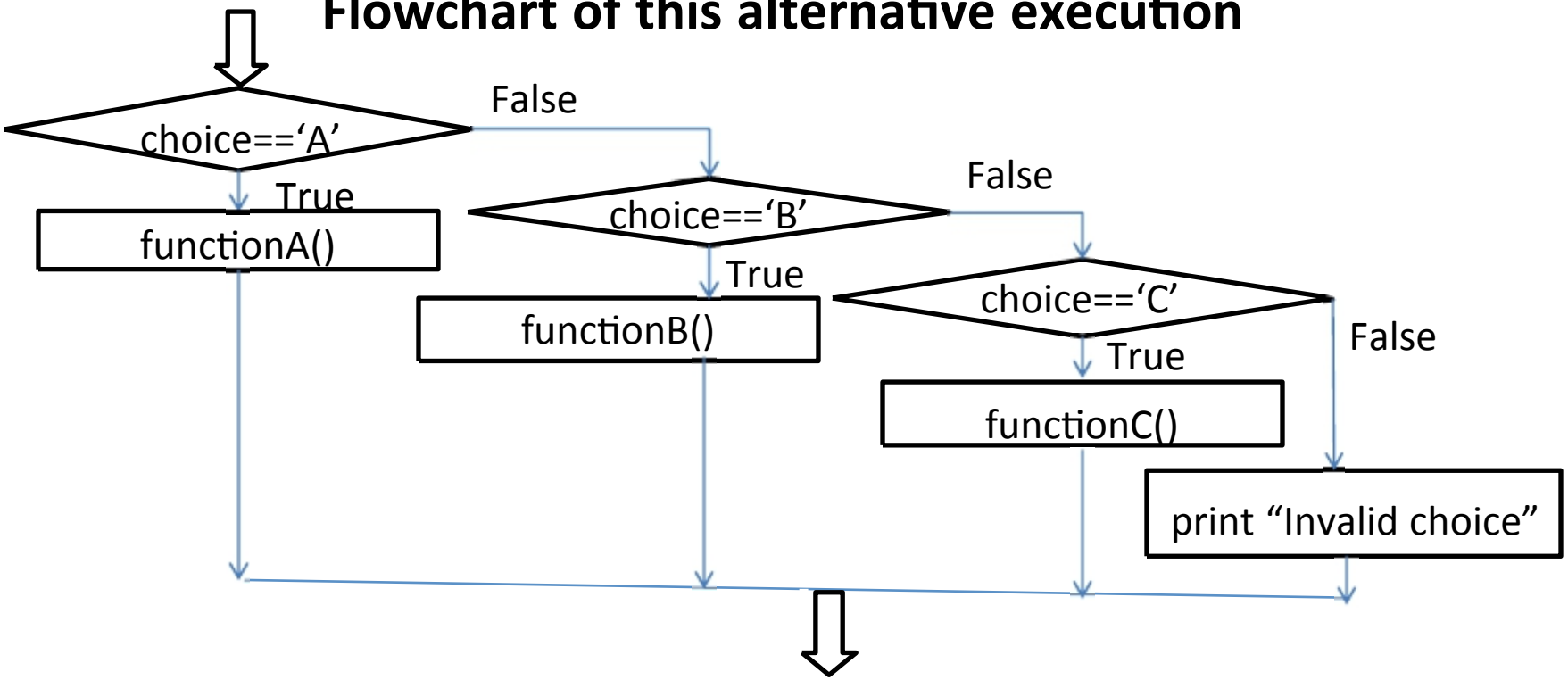
Chained Conditionals

- `if/elif/else`

```
if x < y:
    print x, "is less than", y
elif x > y:
    print x, "is greater than", y
else:
    print x, "and", y, "are equal"
```

```
if choice == "A":
    functionA()
elif choice == "B":
    functionB()
elif choice == "C":
    functionC()
else:
    print "Invalid choice."
```

Flowchart of this alternative execution



Nested Conditionals

```
if x == y:
    print x, "and", y, "are equal"
else:
    if x < y:
        print x, "is less than", y
    else:
        print x, "is greater than", y
```

The above nested conditionals can also be represented as:

```
if x == y:
    print x, "and", y, "are equal"
elif x < y:
    print x, "is less than", y
else:
    print x, "is greater than", y
```

The Return Statement

```
import math
def printLogarithm(x):
    if x <= 0:
        print "The input number must be positive."
        return

    result = math.log(x) #log10 for common logarithm
    print "The natural logarithm of", x, "is", result
```

The `return` statement terminates the execution of the function immediately

The <u>return</u> here does NOT really return anything.

Recursion – the Countdown Example

- It is legal for one function to call another.
- Is it also legal for a function to call itself?

```
def countdown(n):
    if n == 0: #base case
        print "Blastoff!"
    else:
        print n
        countdown(n-1) #general case

countdown(5)
```

Recursion – the Factorial Example

- Mathematical function: factorial definition

```
0!  = 1                (base case)
n!  = n(n - 1)!        (general case)
```

According to the definition, let's calculate 3!

```
3! = 3*2!
   = 3*2*1!
   = 3*2*1*0!
   = 3*2*1*1
3! = 6
```

Recursion – the Factorial Example

```
def factorial(n):
    if n <= 1:
        return 1  # base case
    return n * factorial(n - 1) # general case

print("2! =", factorial(2))
print("3! =", factorial(3))
print("4! =", factorial(4))
print("5! =", factorial(5))
```

Infinite Recursion

- If a recursion **never** reaches a base case, it goes on making recursive calls **forever**, and the program **never terminates**.

```
def countdown(n):
    if n == 0:
        print "Blastoff!"
    else:
        print n
        countdown(n)

countdown(5)
```

Exercise

- Write a recursive Python function that returns the sum of the first **n** positive integers.

```
def sumInt(n):
    if n== 0:
        # write one line here
    else:
        # write one line here
```

Keyboard Input

- More interactive programs
- Get input from the user

raw_input returns what the user typed as **a string**.

```
>>> name = raw_input ("What is your name? ")
What is your name? Harry Potter!
>>> print name
Harry Potter!
```

```
>>> age  = raw_input ("How old are you? ")
How old are you? 19
>>> print "You will be", age+1, "next year!"
```

Summary

- Boolean expression (comparison operators, logical operators)
- Conditional (if) statement (chained or nested when the condition is broken down)
- Recursion does magical work but be careful using it.
- Use raw_input() function to get user input as string.