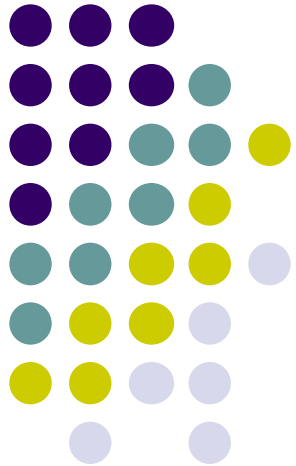
ISYS90088 Introduction to Application Development



Week 4

Boolean and logical operations

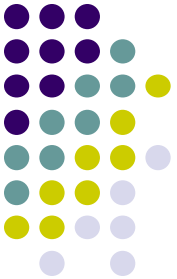
if; if else, nested if, string sequences

Semester 2 , 2017

Dr.Thomas Christy

ISYS90088 sem 2 2017 - Written by Dr Antonette Mendoza
some slides adapted from Fundamentals of Python by Kenneth A. Lambert &local dept. resources

Review



- Binary
- ASCII Char set
- Arithmetic Expressions
- Precedence
- Functions (Math)
- Escape Character

The Boolean Type, Comparisons, and Boolean Expressions

- **Boolean data type** consists of two values: true and false (typically through standard **True/False**)

COMPARISON OPERATOR	MEANING
==	Equals
!=	Not equals
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

[TABLE 3.2] The comparison operators

- Example: 4 != 4 evaluates to False

Logical Operators and Compound Boolean Expressions (continued)

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

A	not A
True	False
False	True

[FIGURE 3.4] The truth tables for **and**, **or**, and **not**

Logical Operators and Compound Boolean Expressions (continued)

- Next example verifies some of the claims made in the previous truth tables:

```
>>> A = True
>>> B = False
>>> A and B
False
>>> A or B
True
>>> not A
False
```

- The logical operators are evaluated after comparisons but before the assignment operator
  - **`not`** has higher precedence than **`and`** and **`or`**

Logical Operators and Compound Boolean Expressions (continued)

TYPE OF OPERATOR	OPERATOR SYMBOL
Exponentiation	`**`
Arithmetic negation	`-`
Multiplication, division, remainder	`*`, `/`, `%`
Addition, subtraction	`+`, `-`
Comparison	`==`, `!=`, `<`, `>`, `<=`, `>=`
Logical negation	`not`
Logical conjunction and disjunction	`and`, `or`
Assignment	`=`

[TABLE 3-4] Operator precedence, from highest to lowest

Logical operation evaluation: Example

- In **(A and B)**, if **A** is false, then so is the expression, and there is no need to evaluate **B**
- In **(A or B)**, if **A** is true, then so is the expression, and there is no need to evaluate **B**

Quiz: 5

Fill in the blanks:

- A compound boolean expression created with a the ---------- operator is true only if both of its sub expressions are true:

Is it the or, and, not?????

The -------- operator takes a boolean expression as its operand and reverses its logical value.

Is it the or, not or and operators?????

Quiz: 5

Fill in the blanks:

- A compound Boolean expression created with a the **AND** operator is true only if both of its sub expressions are true:

Is it the or, and, not?????

The **NOT** operator takes a Boolean expression as its operand and reverses its logical value.

Is it the or, not or and operators?????

Objectives (continued)

- Use logical operators to construct compound Boolean expressions
- Use selection statements to make choices in a program
- String sequences and library functions to manipulate with strings
- Examples

Selection: if and if-else Statements

- **Selection statements** allow a computer to make choices - based on a **condition**

- The **if** statement is used to create a decision structure, which allows a program to have more than one path of execution.

- The **if** statement causes one or more statements to execute only when a Boolean expression is true.

- It is a **control structure** – a logical design that controls the order in which a set of statements execute.

The if – else statement

- The simplest is a **one-way selection** statement (if)
- Also called a **two-way selection** statement (if-else)
- The condition in the if-else statement must be a Boolean expression – that is, an expression that evaluates to either true or false

The Boolean Type, Comparisons, and Boolean Expressions

- Boolean data type consists of two values: true and false (typically through standard True/False)

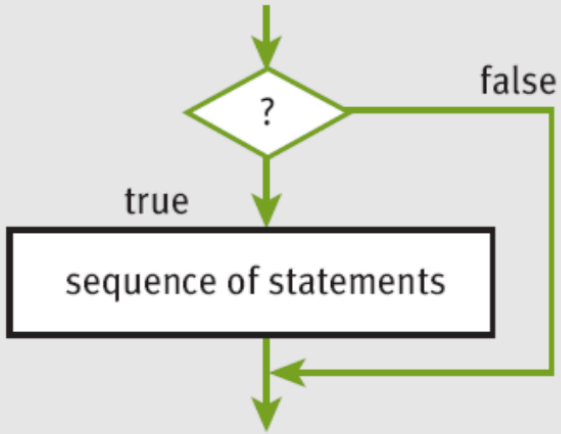
COMPARISON OPERATOR	MEANING
==	Equals
!=	Not equals
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

- Example: 4 != 4 evaluates to False

One-Way Selection Statements

- Simplest form of selection is the ***if* statement**

```
if <condition>:
    <sequence of statements>
```




```
>>>x = -2
>>>if x < 0:
        x = abs(x)
>>>print(x)
```

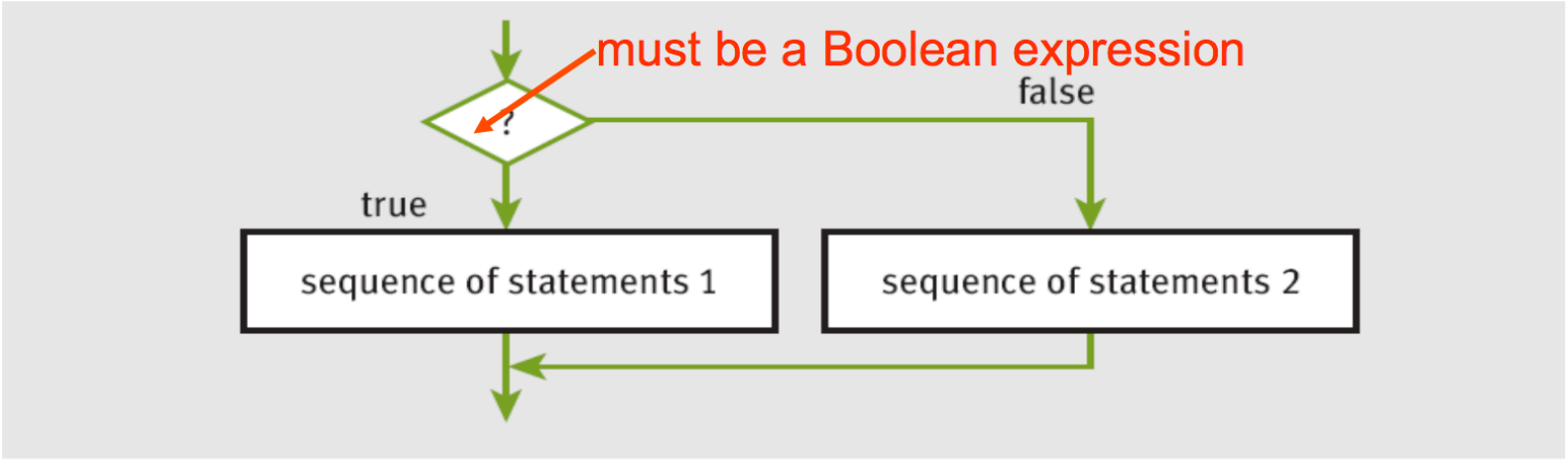
if-else Statements

- The two possible actions each consist of a sequence of statements
- Each sequence must be indented at least one space beyond the symbols if and else.

Syntax:

```
if <condition>:
    <sequence of statements-1>
else:
    <sequence of statements-2>
```

if-else Statements (continued)



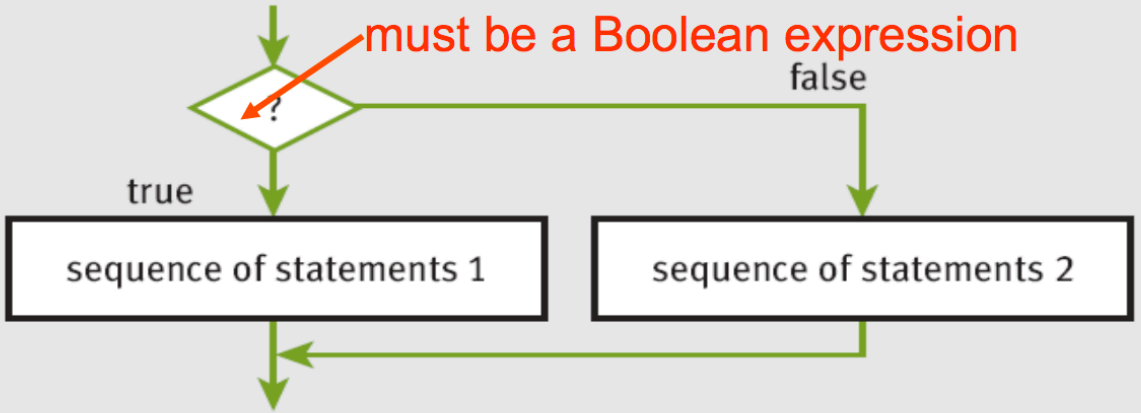
Lets look at an example…

if-else Statements (continued)

Write a program to accepts two positive integers and print out the maximum and the minimum of the two input numbers - using max and min functions from **the math library**

(example 4)

if-else Statements (continued)




```
first = int(input("Enter the first number: "))
second = int(input("Enter the second number: "))
if first > second:
    maximum = first
    minimum = second
else:
    maximum = second
    minimum = first
print("Maximum:", maximum)
print("Minimum:", minimum)
```

Multi-way if Statements

- A program may be faced with testing conditions that entail more than two alternative courses of action

LETTER GRADE	RANGE OF NUMERIC GRADES
A	All grades above 89
B	All grades above 79 and below 90
C	All grades above 69 and below 80
F	All grades below 70

- Can be described in code by a **multi-way selection statement**

Multi-way if Statements (continued)

- At most one of the indented blocks will run.
- The conditions are tried in order until one is found that is True. The associated block of code is run and any remaining conditions and blocks are skipped.
- If none of the conditions are True but there is an else block, then Python runs the else block.

Syntax:

```
if <condition-1>:
    <sequence of statements-1>

elif <condition-n>:
    <sequence of statements-n>
else:
    <default sequence of statements>
```

Logical Operators and Compound Boolean Expressions with if-else

- Often a course of action must be taken if either of two conditions is true: Below are two approaches

```
number = int(input("Enter the numeric grade: "))
if number > 100:
    print("Error: grade must be between 100 and 0")
elif number < 0:
    print("Error: grade must be between 100 and 0")
else:
    # The code to compute and print the result goes here
```

```
number = int(input("Enter the numeric grade: "))
if number > 100 or number < 0:
    print("Error: grade must be between 100 and 0")
else:
    # The code to compute and print the result goes here
```

Logical Operators and Compound Boolean Expressions with if-else

- Often a course of action must be taken if either of two conditions is true: Below are two approaches

```
number = int(input("Enter the numeric grade: "))
if number > 100:
    print("Error: grade must be between 100 and 0")
elif number < 0:
    print("Error: grade must be between 100 and 0")
else:
    # The code to compute and print the result goes here
```

```
number = int(input("Enter the numeric grade: "))
if number > 100 or number < 0:
    print("Error: grade must be between 100 and 0")
else:
    # The code to compute and print the result goes here
```

Example: Multi-way if Statements (continued)

Write a program to convert numeric grades of your ISYS90088 test scores to letter grades. The grades are as follows:

- H1 all grades 80 and above(80 - 100)
- H2A all grades equal to and above 75 but below 80 (75-79)
- H2B all grades between grades(70-74)
- P all grades between (50 -69)
- F all grades below 50

(examples 5, 6)

Testing Selection Statements

- Tips:
  - Make sure that all of the possible branches or alternatives in a selection statement are exercised
  - After testing all of the actions, examine all of the conditions
  - Test conditions that contain compound Boolean expressions using data that produce all of the possible combinations of values of the operands

Objectives: strings

- Access individual characters in a string
- Retrieve a substring from a string
- Search for a substring in a string
- Using library for string manipulations

Accessing Characters and Substrings in Strings

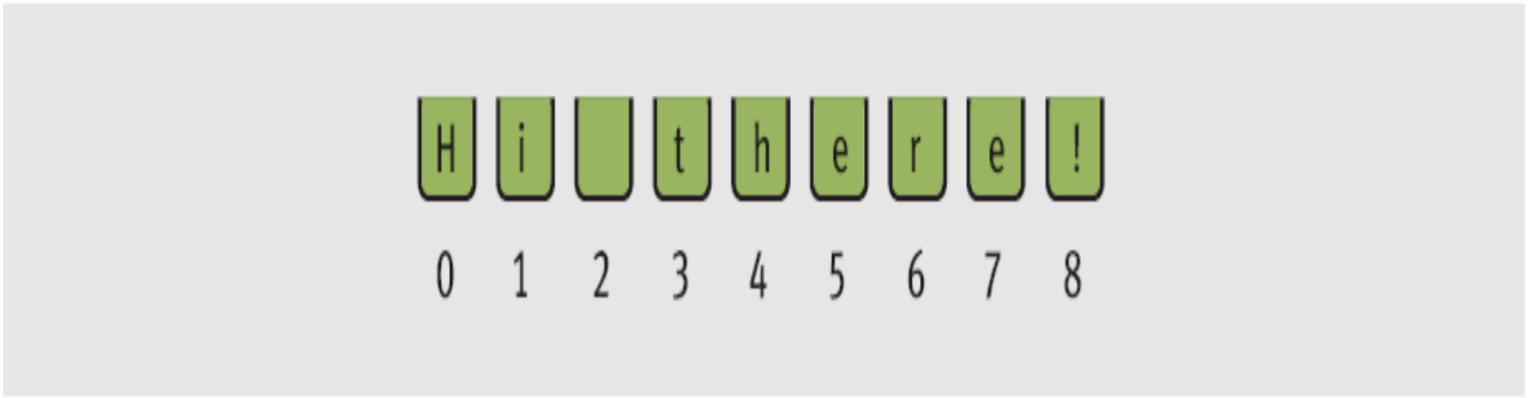
- In this section, we examine the internal structure of a string more closely
- You will learn how to extract portions of a string called **substrings**

The Structure of Strings

- **Data structure** – it is an organized way of storing and representing data so that data can be inserted and accessed - data structure consists of smaller pieces of data.
- A string is a data structure.
- The string is a sequence of zero or more characters
- A string is an **immutable** data structure. That means, its internal data elements, the characters, can be accessed, but the structure itself cannot be modified.

The Structure of Strings

- We sometimes might want to access or inspect characters at a given position without needing to visit the entire string (part of a string – substring)
- The string is arranged as shown. There is **an index** that helps us step thru' or inspect a position



Indexing and the Subscript Operator

```
<a string>[<an integer expression>]
```

index is usually in range [0,length of string – 1]; can be negative

- Examples:

```
>>> name = "Alan Turing"
>>> name[0]                          # Examine the first character
'A'
>>> name[3]                          # Examine the fourth character
'n'
>>> name[len(name)]                  # Oops! An index error!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> name[len(name) - 1]              # Examine the last character
'g'
>>> name[-1]                         # Shorthand for the last one
'g'
```

Strings indexing

- For example: String's length - Number of characters it contains (0+). Len is a library function that allows us to do some manipulation with strings.

Example:

>>>s = "example"

>>>s[2]

'a'

```
>>> len("Hi there!")
9
>>> len("")
0
```

Negative Indexing

- We have seen what happens when the index is too large. What happens if the index is less than 0? Does it give us an error? NO

```
>>>s = "The number is 42."
>>>print(s[-1])
>>>print(s[-2])
>>>print(s[-3])
>>>print(s[-17])
```

- Negative indices work from the end of the string, so **-1** indexes the last character, which is . and **-17** indexes the 17th last character, which is *T* (actually the first character of the string).
- Note the negative indexes are one-offset (i.e. start from **-1**) while the positive indexes are zero-offset (i.e. start from **0**).
- Example 7

Slicing for Substrings

- Python's subscript operator can be used to obtain a substring through a process called **slicing**
  - Place a colon (**:**) in the subscript; an integer value can appear on either side of the colon
- For example, the following code accesses the substring of the string **"the example is on slicing"** starting at index **2**, up to (but not including) index **10**.

```
>>>s = "the example is on slicing"
>>>print (s[2:10])
e exampl
```

Accessing Substrings (Slicing)

- The notation **2:10** is known as a slice.
- Remember that a slice starts at the first index but finishes one before the end index. This is consistent with indexing: indexing also starts from zero and goes up to one before the length of the string.
- You can see this by slicing with the value of **len**:

```
>>>s = "the example is on slicing"
>>print (len(s))
>>print (s[0:len(s)])
```

More on Slicing: example

- You can also slice with negative indices. The same basic rule of starting from the start index and stopping one before the end index applies:

```
>>>s = ”testing slicing 101"
>>>print(s[4:-7])
>>>print(s[-7:-1])
>>>print(s[-6:len(s)])
```

Solution:
????

Example 7

Changing the Step Size and Direction

- You can specify a third number which indicates how much to step through the list by.
- For example, if you want every second element you can do this:

```
>>>s = "abcdef"
>>>print(s[::2])
```

- Or you can specify a range with a step:

```
>>>s = "abcdef"
>>>print(s[0:3:2])
```

(Example 8)

Changing the Step Size and Direction

- If this third number is -1 it changes the direction you are slicing in:

```
>>>s = "abcdef"
>>>print(s[2::-1])
>>>print(s[2:0:-1])
>>>print(s[-4:-6:-1])
cba
cb
cb
```

- Note that the direction of the indices must be changed also. If there is nothing in between the indices, an empty string is returned (unlike indexing beyond the ends of the string, which led to an error):

```
>>>s = "abcdef"
>>>print(s[0:2:-1])
```

Testing for a Substring with the in Operator

- When used with strings, the left operand of **in** is a target substring and the right operand is the string to be searched
  - Returns **True** if target string is somewhere in search string, or **False** otherwise

- Example:

```
>>>list = ["ant", "dog", "cat", "rat", "horse"]
>>>"ant" in list
True
```

String Methods

- Python includes a set of string operations called **methods** that make tasks like counting the words in a single sentence easy
- dir(str) or help(str) will give you the entire list of string methods or library functions.
- A method behaves like a function, but has a slightly different syntax
  - A method is always called with a given data value called an **object**

```
<an object>.<method name>(<argument-1>, …, <argument-n>)
```

String Methods (continued)

- Methods can expect arguments and return values
- A method knows about the internal state of the object with which it is called
- In Python, all data values are objects

String Methods (continued)

STRING METHOD	WHAT IT DOES
`s.center(width)`	Returns a copy of `s` centered within the given number of columns.
`s.count(sub [, start [, end]])`	Returns the number of non-overlapping occurrences of substring `sub` in `s`. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.endswith(sub)`	Returns `True` if `s` ends with `sub` or `False` otherwise.
`s.find(sub [, start [, end]])`	Returns the lowest index in `s` where substring `sub` is found. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.isalpha()`	Returns `True` if `s` contains only letters or `False` otherwise.
`s.isdigit()`	Returns `True` if `s` contains only digits or `False` otherwise.

String Methods (continued)

STRING METHOD	WHAT IT DOES
`s.join(sequence)`	Returns a string that is the concatenation of the strings in the sequence. The separator between elements is `s`.
`s.lower()`	Returns a copy of `s` converted to lowercase.
`s.replace(old, new [, count])`	Returns a copy of `s` with all occurrences of substring `old` replaced by `new`. If the optional argument `count` is given, only the first `count` occurrences are replaced.
`s.split([sep])`	Returns a list of the words in `s`, using `sep` as the delimiter string. If `sep` is not specified, any whitespace string is a separator.
`s.startswith(sub)`	Returns `True` if `s` starts with `sub` or `False` otherwise.
`s.strip([aString])`	Returns a copy of `s` with leading and trailing whitespace (tabs, spaces, newlines) removed. If `aString` is given, remove characters in `aString` instead.
`s.upper()`	Returns a copy of `s` converted to uppercase.

String Methods (continued)

```
>>> s = "Hi there!"
>>> len(s)
9
>>> s.center(11)
' Hi there! '
>>> s.count('e')
2
>>> s.endswith("there!")
True
>>> s.startswith("Hi")
True
>>> s.find('the')
3
>>> s.isalpha()
False
>>> 'abc'.isalpha()
True
>>> "326".isdigit()
True
>>> words = s.split()
>>> words
['Hi', 'there!']
>>> "".join(words)
'Hithere!'
```

String Methods (continued)

```
>>> " ".join(words)
'Hi there!'
>>> s.lower()
'hi there!'
>>> s.upper()
'HI THERE!'
>>> s.replace('i', 'o')
'Ho there!'
>>> " Hi there! ".strip()
'Hi there!'
>>>
```

An example for split – an intro to make things interesting - tuple

```
s = '2 3 4'
(a, b, c) = s.split()
print(a)
print(b)
print(c)
```

Example : program – try this one!

#Imagine you are writing a title for a business report in your organization. The title should be not more that 10 words. Write a program that accepts from the user the title and :

i. Makes sure the title is in upper case. If its not, your program converts the title to all upper case;

ii. counts the number of words in the title. If the number of words is >10, your program will inform the user that the title must be not more that 10 words.

(example 9)