CSI31 Lecture 6

Topics:

- 3.5 Limitations of Computer Arithmetic
- 7.1 Simple Decisions
- 7.2 Two-way Decisions

3.5 Limitations of Computer Arithmetic

Recall our *factorial function* **n! :**
sometimes it is suggested that **!** is there for a reason - meaning that this function grows very rapidly.

For example, 50! = 30414093201713378043612608166064768844377641568960512000000000000

3.5 Limitations of Computer Arithmetic

Recall our *factorial function* **n! :**
sometimes it is suggested that **!** is there for a reason - meaning that this function grows very rapidly.

For example, 50! = 30414093201713378043612608166064768844377641568960512000000000000

recent versions of Python have no difficulty with this calculation.
Other versions of Python as well as other programming languages (like C++, Java) would not fare as well.

For example, in Java, if we write a similar program,
13! = 1, 932, 053, 504, but if we check it:
13! is actually 6, 227, 020, 800

3.5 Limitations of Computer Arithmetic

It is important to keep in mind, that computer representations of numbers (the actual data types) do not always behave exactly like the numbers that they stand for.

Java program uses the underlying **int** data type, and relies on the computer addition operation for **ints**.

There are infinitely many integers, but only a **finite range of ints**.

3.5 Limitations of Computer Arithmetic

The number of bits a particular computer uses to represent an **int** depends on the design of the CPU.

with two bits

bit 2	bit 1
0	0
0	1
1	0
1	1

we can represent 4 things (2^2)

bit 3	bit 2	bit 1
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

with three bits we can represent 8 things:
$2^3 = 8$

3.5 Limitations of Computer Arithmetic

The number of bits a particular computer uses to represent an **int** depends on the design of the CPU.

Typical PCs today use 32 or 64 bits.

3.5 Limitations of Computer Arithmetic

The number of bits a particular computer uses to represent an **int** depends on the design of the CPU.

Typical PCs today use 32 or 64 bits.
Therefore, for a 32 bit CPU, there are 2^{32} possible values, which are centered at 0, to represent the range of positive and negative integers.

$$\frac{2^{32}}{2} = 2^{31} = 2,147,483,647$$

$[-2^{31}, 2^{31}-1]$

3.5 Limitations of Computer Arithmetic

The number of bits a particular computer uses to represent an **int** depends on the design of the CPU.

Typical PCs today use 32 or 64 bits.
Therefore, for a 32 bit CPU, there are 2^{32} possible values, which are centered at 0, to represent the range of positive and negative integers.

$$\frac{2^{32}}{2} = 2^{31} = 2,147,483,647$$

$[-2^{31}, 2^{31}-1]$

$12! \leq 2^{31} \leq 13!$ hence Java program is fine for calculating factorials up to 12, but after that the representation «overflows» and the results are garbage.

3.5 Limitations of Computer Arithmetic

Why does the modern Python program seems to work quite well computing with large integers?

3.5 Limitations of Computer Arithmetic

Why does the modern Python program seems to work quite well computing with large integers?

- Python's **int** is not a fixed size.
It expands to accomodate whatever value it holds.
The only limit is the amount of memory the computer has available to it.

3.5 Limitations of Computer Arithmetic

Why does the modern Python program seems to work quite well computing with large integers?

- Python's **int** is not a fixed size.
It expands to accomodate whatever value it holds.
The only limit is the amount of memory the computer has available to it.

Of course, in order to perform operations on larger numbers, Python has to break down operations into smaller units that the computer hardware is able to handle.

7.1 Simple Decisions (If-statement)

Syntax of the *if-statement*:

```
if <condition>:
   body
```

Example:

```
if t>90:
   print(''Heat Warning!'')
```

7.1 Simple Decisions (If-statement)

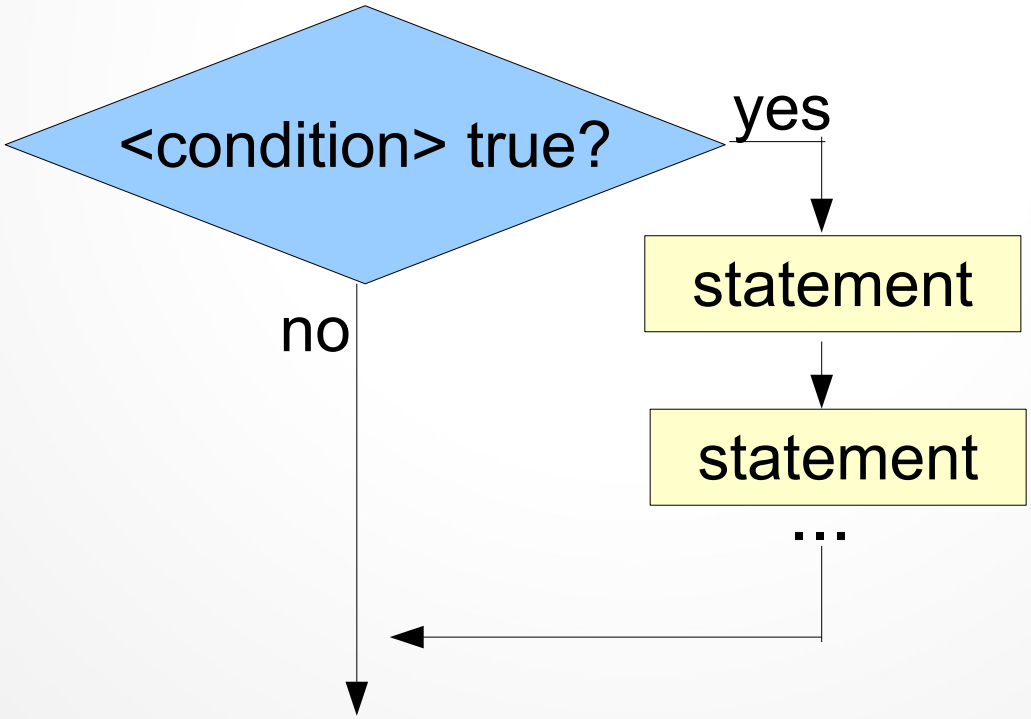
Syntax of the *if-statement*:

```
if <condition>:
   body
```

Example:

```
if t>90:
   print(''Heat Warning!'')
```

Control flow of simple if-statement:



7.1 Simple Decisions (If-statement)

Syntax of the *if-statement*:

```
if <condition>:
   body
```

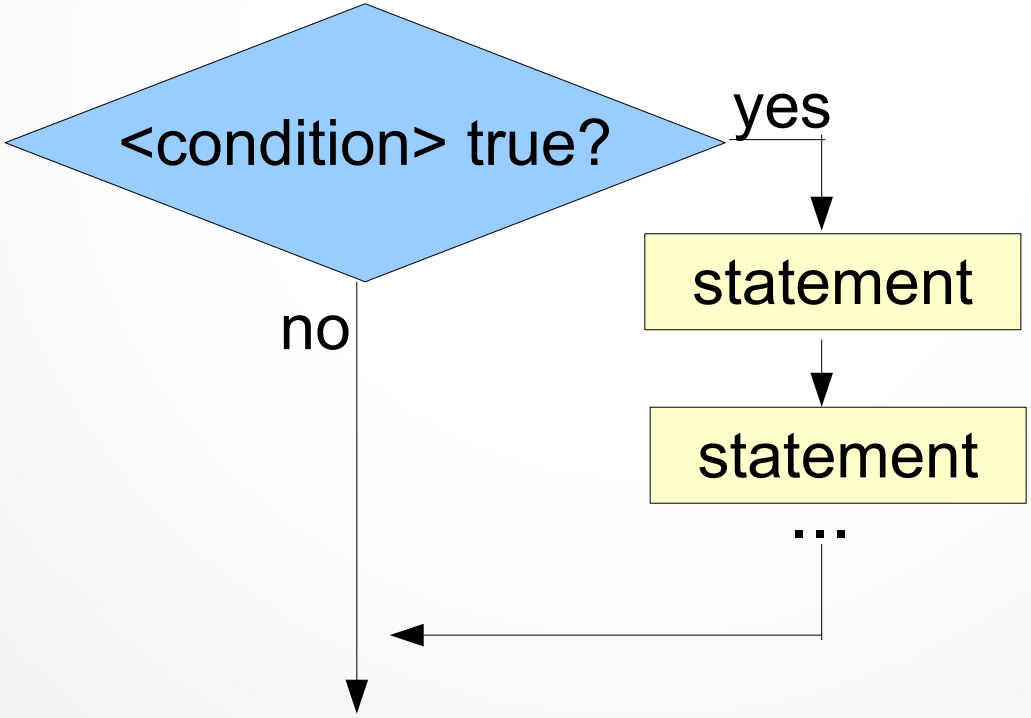
Example:

condition

```
if t>90:
   print(''Heat Warning!'')
```

body

Control flow of simple if-statement:



7.1 Simple Decisions (If-statement)

Conditions:

Try to input the following commands in the interactive window (Python's shell):

```
>>> 3*7 > 23

>>> 23 <= 4*6*0

>>> 5 == 5

>>> 5=5
```

7.1 Simple Decisions (If-statement)

Example: *Online Wine Store*
Let's write a simple program of an Online Wine Store.
Our store will offer 6 types of wine: *Merlot, Sauvignon, Ice Wine, Pinot Noir, Chardonnay*, and *Cabernet*.

A customer will be able to choose only one type of wine, but any number of bottles of the selected wine.

At the end of selection we will simply notify the user that the order is forwarded to the checkout.

7.1 Simple Decisions (If-statement)

Example: *Online Wine Store*

Thoughts: what control structure can/should we use?
loops or simple decisions?

Determining Specification:
Input: type of wine, number of bottles
Output: confirmation of the selection and forwarding to the checkout
check age restrictions

7.1 Simple Decisions (If-statement)

Example: *Online Wine Store*

Design/Algorithm:
get age of the customer,
check the age restrictions
get type of the wine from the user (the user will be given a list of "number choices")
get the number of bottles
display confirmation of the selection
forward to the checkout

see **simpleDecision.py**

7.1 Simple Decisions (If-statement)

Example: *Online Wine Store*

Possible modification: allow the customer to select up to 4 types of wine.

see **simpleDecision_mod.py**

Yet another possible modification: we need to take care of:
- cases when the choice is out of the offered range of types of wine, and
- we don't need to print anything when customer selects 7 (none of the wines)

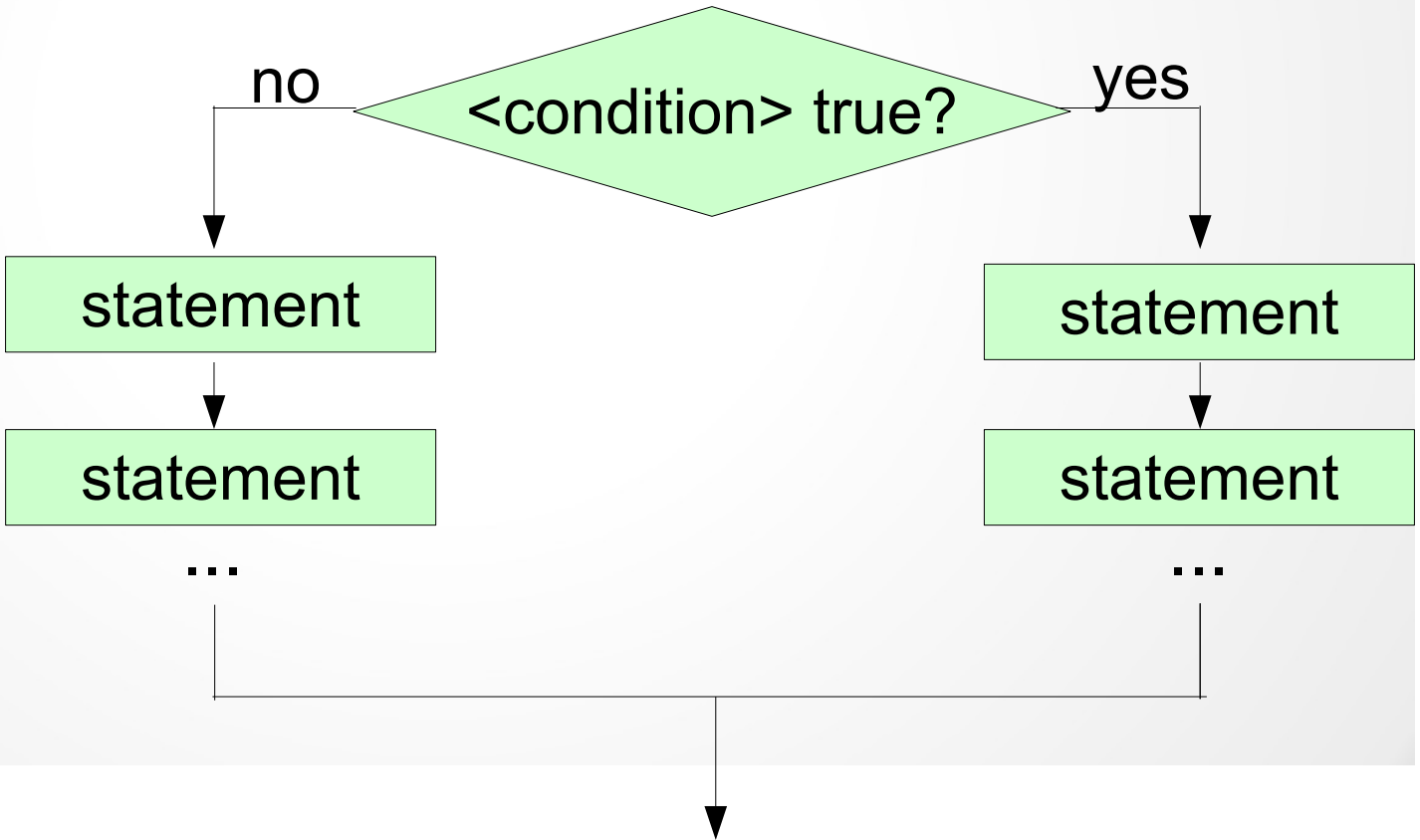
see **simpleDecision_mod2.py**

7.2 Two-way Decisions (if-else statement)

Syntax of the *if-else statement*:

```
if <condition>:
   <statements>
else:
   <statements>
```

Control flow of a two-way decision `if-else` statement:



7.2 Two-way Decisions (if-else statement)

Example: *Programming exercise 1*
Many companies pay time-and-a-half for any hours worked above 40 in a given week. Write a program to input the number of hours worked and the hourly rate and calculate the total wages for the week.

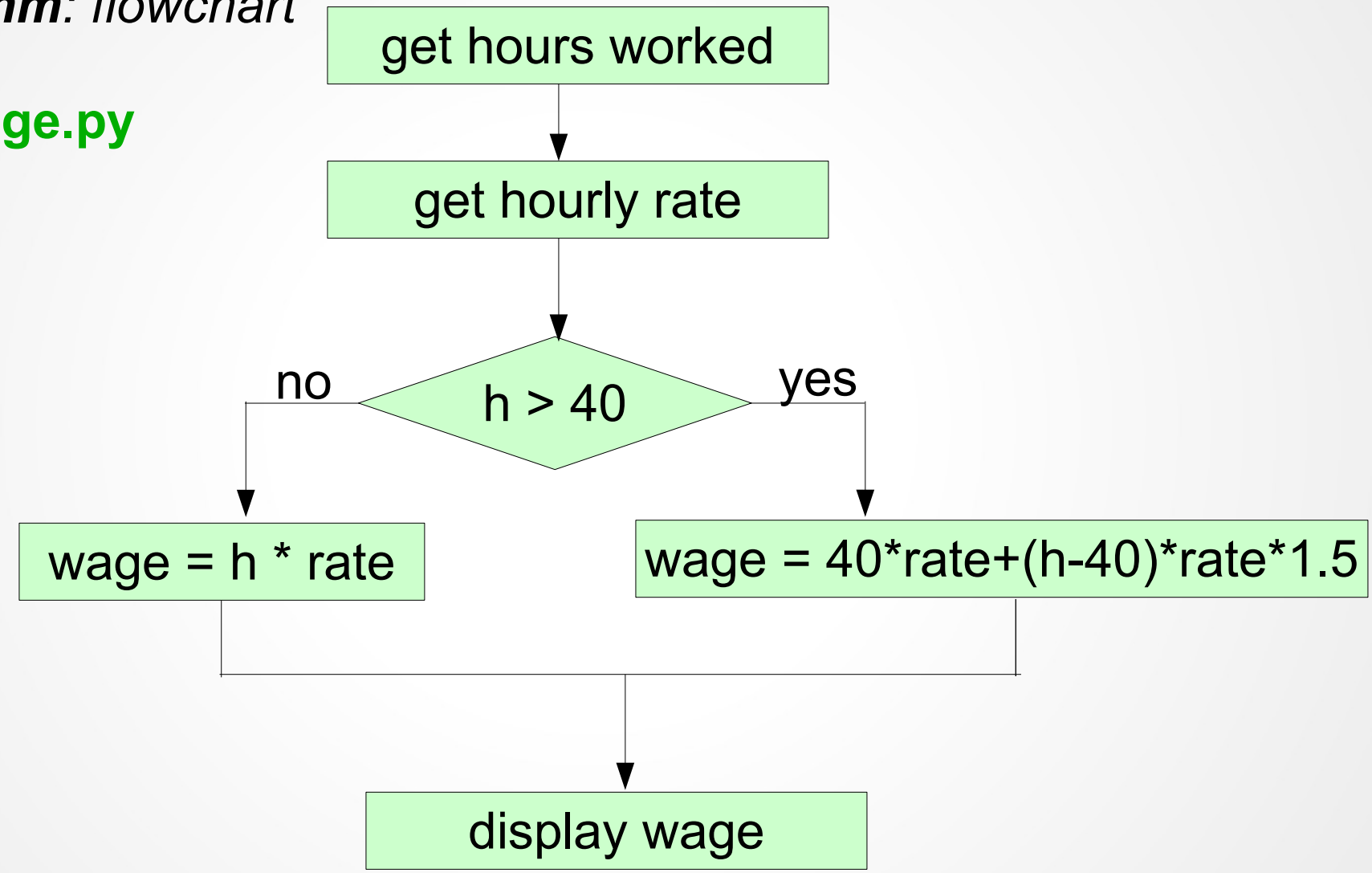
Software design:
input: the number of hours worked in a given week (*h*)
hourly rate (*rate*)
output: total wage for the week (*f(h,rate)*)
relation: calculate the wage using

$$f(h, rate) = \begin{cases} h * rate, & \text{if } h \le 40 \\ 40 * rate + (h - 40) * rate * 1.5, & \text{if } h > 40 \end{cases}$$

7.2 Two-way Decisions (if-else statement)

***Algorithm**: flowchart*

see **wage.py**



7.2 Two-way Decisions (if-else statement)

Testing/Debugging:
do a thorough testing

We can use a table of calculations that we performed ourselves:

hours	hourly rate	calculation	wage
30	$12	30*$12 = $360	$360
41	$10	40*$10 + 1*$10*1.5 = $415	$415
29	$20	29*$20 = $580	$580
48	$18	40*$18 + 8*$18*1.5 = $936	$936
51	$11	40*$11 + 11*$11*1.5 = $621.50	$621.50
43	$12.60	40*$12.60 + 3*$12.60*1.5 = $560.70	$560.70

7.1.2 Forming Simple Conditions

How does a condition looks exactly?

syntax: `<expr> <relop> <expr>`
`<expr>` - expression
`<relop>` - relational operator

Python	Mathmatics	Meaning
<	<	*less than*
<=	≤	*less than or equal to*
==	=	*equal to*
>=		*greater than or equal to*
>	>	*greater than*
!=	≠	*not equal to*

there are six relational operators in Python

7.1.2 Forming Simple Conditions

Conditions are a type of expressions, called *Boolean* expression.

When a Boolean expression is evaluated, it produces a value of either:
true (the condition holds) or
false (it doesn't hold).

In some languages, 1 and 0 (*type int*) are used to represent true and false, correspondingly.

In Python, Boolean expressions are of type `bool`.

7.1.2 Forming Simple Conditions

Type in the following in Python's shell:

```
>>> 12 < 10
False
>>> 2*6 == 1*6
False
>>> 1 > 7 or (-3)**2>0
True
>>> 1 > 7 and (-3)**2>0
False
>>> not 1 > 7
True
```