CSI31 Lecture 3

Topics:

2.1 The Software Development Process
2.3 Elements of Program
2.4 Output Statements
2.5 Asssignment Statements

2.1 The Software Development Process

Stages of the process of creating a program:

Analyze the problem

Determine specifications

Create a design

Implement the design

Test/Debug the program

Maintain the program

2.1 The Software Development Process

Stages of the process of creating a program:

Analyze the problem

- figure out exactly what is the problem to be solved

Determine specifications

Create a design

Implement the design

Test/Debug the program

Maintain the program

2.1 The Software Development Process

Stages of the process of creating a program:

Analyze the problem

Determine specifications - describe exactly what your program will do (not *how* it will work, but *what it will accomplish*)
(for simple programs: what is the input/output, how they relate to each other)

Create a design

Implement the design

Test/Debug the program

Maintain the program

2.1 The Software Development Process

Stages of the process of creating a program:

Analyze the problem

Determine specifications

Create a design - formulate overall structure of the program (how the program will work)
- algorithms are usually written in *pseudocodes*

Implement the design

Test/Debug the program

Maintain the program

2.1 The Software Development Process

Stages of the process of creating a program:

Analyze the problem

Determine specifications

Create a design

Implement the design

- translate the algorithm into a computer language

Test/Debug the program

Maintain the program

2.1 The Software Development Process

Stages of the process of creating a program:

Analyze the problem

Determine specifications

Create a design

Implement the design

Test/Debug the program

- see if it works as expected (run on as many different inputs as you can; you should try everything you can think of that might «break» your program — *testing*)
- check for errors (*bugs*) – fix them – *debugging*

Maintain the program

2.1 The Software Development Process

Stages of the process of creating a program:

Analyze the problem

Determine specifications

Create a design

Implement the design

Test/Debug the program

Maintain the program - continue developing/updating the program in response to the needs of your users.

2.1 The Software Development Process

Example

Let's go through the steps of the software development process with the following example:

I'd like to write a program that measures the area of a rectangular room. I assume that the input data will be in inches and I'd like to output result in square meters.

2.1 The Software Development Process

Example

I'd like to write a program that measures the area of a rectangular room. I assume that the input data will be in inches and I'd like to output result in square meters.

Analisys: (analyze the problem)

I need a program that measures the area of a rectangular room.
I'll be given length and width. In inches.
The output should be the area in square meters.

2.1 The Software Development Process

Example

I'd like to write a program that measures the area of a rectangular room. I assume that the input data will be in inches and I'd like to output result in square meters.

Analisys: (analyze the problem)

I need a program that measures the area of a rectangular room.
I'll be given length and width. In inches.
The output should be the area in square meters.

Determine specifications: program will:

- notify the user of what it can do,
- ask to input the length in inches
- ask to input the width in inches
- calculate the area
- output the result

2.1 The Software Development Process

Example

I'd like to write a program that measures the area of a rectangular room. I assume that the input data will be in inches and I'd like to output result in square meters.

Design an algorithm:

- input the length of the room
- input the width of the room
- calculate A= (W * 2.54 * 0.01) * (L * 2.54 * 0.01)
- output area

2.1 The Software Development Process

Example

I'd like to write a program that measures the area of a rectangular room. I assume that the input data will be in inches and I'd like to output result in square meters.

Implementation:

I google'ed for the conversion from inches to meters, and found the following:

1 meter = 100 centimeters (or 1 centimeter is 0.01 meters)

1 inch = 2.54 centimeters

Thus if I have *n* inches, it will be *n * 2.54* centimeters, and *n * 2.54 * 0.01* meters.

The formula for the area of the rectangle is *A = W * L*.

So the final formula for the area in meters is

*A = (W * 2.54 * 0.01) * (L * 2.54 * 0.01)*

see the program area.py

2.1 The Software Development Process

Example

I'd like to write a program that measures the area of a rectangular room. I assume that the input data will be in inches and I'd like to output result in square meters.

Test/Debug the program:
test on several inputs (0,0), (100,100), (1000,1000), (4,5)

Maintenance:
not needed right now (possibly in the future)

2.3 Elements of Programs

Names (identifiers)

we give names to modules (files), to functions, to variables. Technically these names are called *identifiers*.

Python has some **rules** about how *identifiers* are formed:
must begin with a letter or underscore ("_"), may be followed by any sequence of letters, digits or underscores, but no spaces, points, commas, ...

legal names:	illegal names:
counter	x.y
x2	net pay
x2_y	10monkeys
ToGoThere	_my-counter
_234brush	

2.3 Elements of Programs

Names (identifiers)

! Identifiers are case-sensitive, thus Counter, counter, counTer, COUNTER are different names

! Some identifiers are part of the Python itself (they are *reserved words*), cannot be used as ordinary identifiers (see Table 2.1 on page 32)

2.3 Elements of Programs

Expressions

the fragment of a code that produce or calculate new data is called *expression*.

A simplest kind of expression is *literal*:

```
5   in  x = 5                2.4    in  y = 2.4
True in flag = True
Hello   in word = ''Hello'' (string literal)
```

more complex and interesting expressions are constructed by combining simpler expressions with operators, and variables.

Operators for numbers: *, +, -, /, **. (2 ** 4 = 2^4)

example: (((x+3) * (y-2)) ** 2 + 1023) / 5.4

2.3 Elements of Programs

Expressions

Expressions are the fragments of a program that produce data, and are composed of *literals*, *variables* and *operators*.

For more information on operators see
Python documentation -> Language Reference -> Expressions

If Python cannot find a value – it reports a NameError.
(in the interactive window try to type in :
>>> print(x)
and see what will be the response)

2.4 Output Statements

Command `print`

Let's take a look at printing statements that display information on the screen:

Syntax of the `print` statements:

`print()` - will produce a blank line of output

`print(<expr>)`

`print(<expr_1>, <expr_2>, ..., <expr_n>)` - sequence of expressions

2.5 Assignment Statements

The basic assignment statement's form:

```
<variable> = <expr>
```

identifier — expression

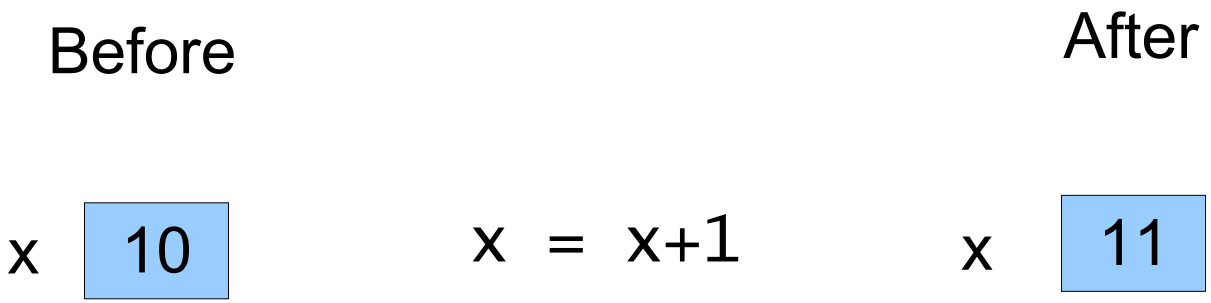
example: `x = 9.8 * x * (32 + x)`

A variable can be assigned values many times.

```
x = 1000
x = 4+15
x = 4/5
```

- it always returns the value of the most recent assignment (each time variable switches to refer to the new value – Python works this way)

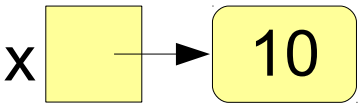
2.5 Assignment Statements



Variable as box

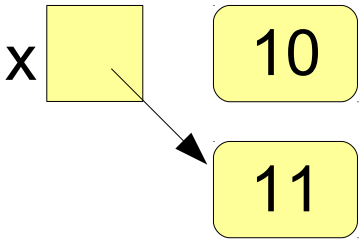
2.5 Assignment Statements

Before



x = x+1

After



Variable as sticky note (Python)

Garbage collection

When a value is no longer referred to by **any** variable, it is no longer useful. Python will automatically clear these values out of memory – *garbage collection*.

2.5 Assignment Statements

Simultaneous Assignment

- an alternative form of the assignment statement that allows to calculate several values at the same time

syntax:
<var_1>, <var_2>, ... <var_n> = <expr_1>, <expr_2>, ..., <expr_n>

semantics: tells the Python to evaluate all the expressions on the right-hand side and then assign these values to the corresponding variables named on the left-hand side.

Example: `sum, diff = 4+3, 4-3`
sum is 7, diff is 1

2.5 Assignment Statements

Simultaneous Assignment

Simultaneous assignment can be used for quick swapping of values.

Example: Assume that x = 4, y = 6 – and we want to swap their values.
We will type in:

```
x, y = y, x
```

Otherwise we will have to do the following:

```
temp = y
y = x
x = temp
```

Word of caution: do not do this in C++

Input Statements

Input statement is used to get some information from the user of the program and store into a variable.
(for this we use an assignment statement along with a special expression called input)

syntax:
<variable> = input(<prompt>) (assignment of a string of characters)
<variable> = eval(input(<prompt>)) (assignment of a number)

`prompt` is an expression that serves to prompt the user for input
(almost always a string literal)

Input Statement with multiple assignment

another example: get three values from the user and find the average of those numbers.

```
def main():
   print(''let's find the average of three
            numbers'')
   x,y,z = eval(input(''Input three numbers
                          separated by a coma: ''))
   average = (x+y+z) / 3
   print(''The average of those numbers is '',
          average)

main()
```

Danger of eval

Take a look at the following interaction with the Python interpreter:

```
>>> ans = eval(input("Enter an expression:"))
Enter an expression: 12-6*2-6/2
>>> print(ans)
-3
```

Danger: when we evaluate user's input, we are essentially allowing the user to enter a portion of our program!
Someone could exploit this ability to enter malicious instructions (capture private information or delete files on the computer).
In computer security it is called a *code injection attack*.