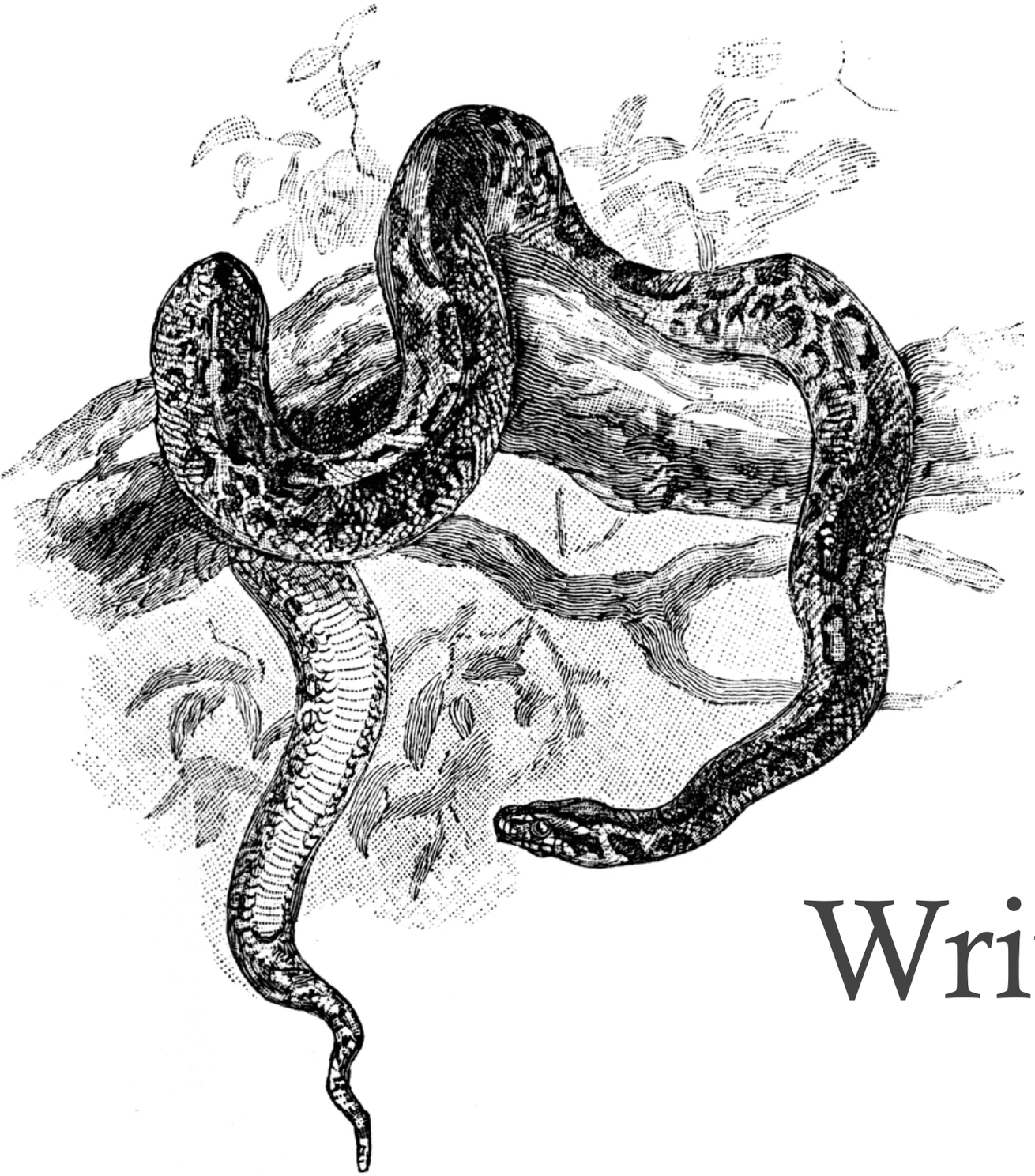
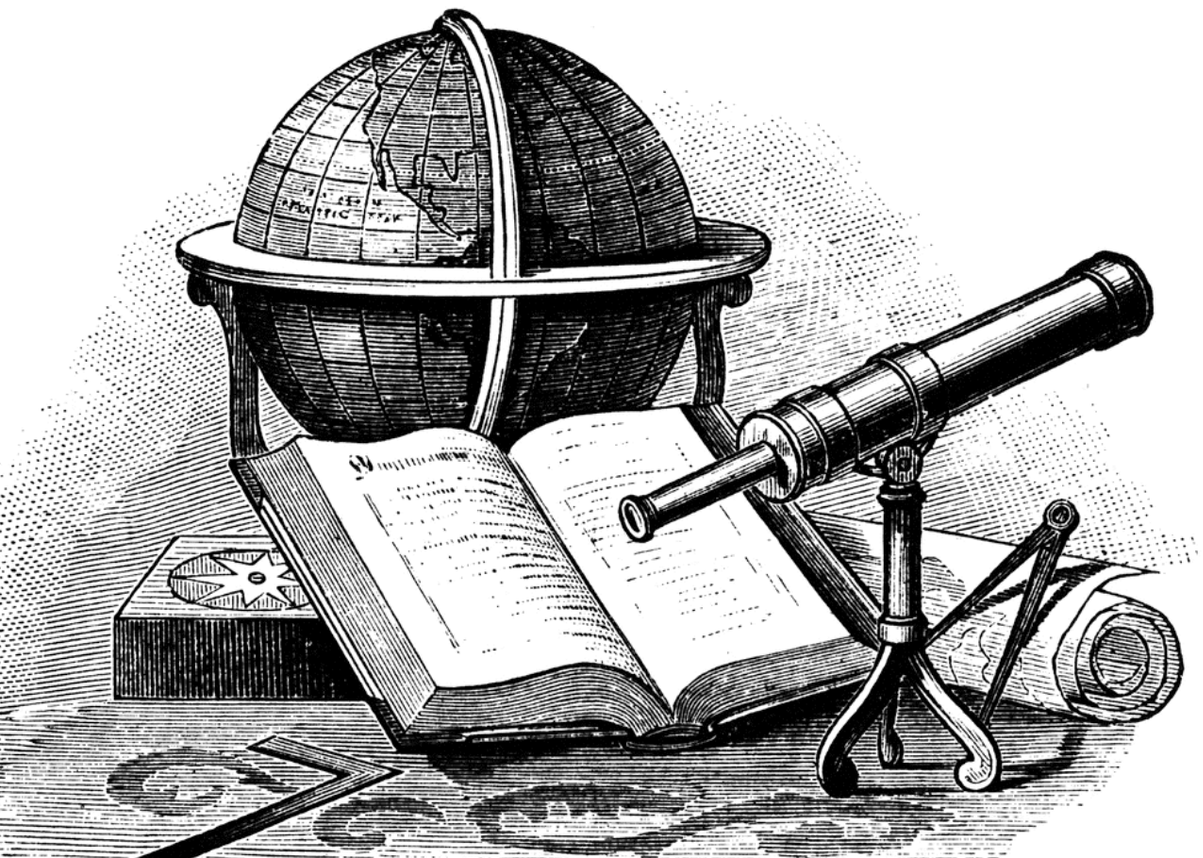


Lecture

Writing Programs

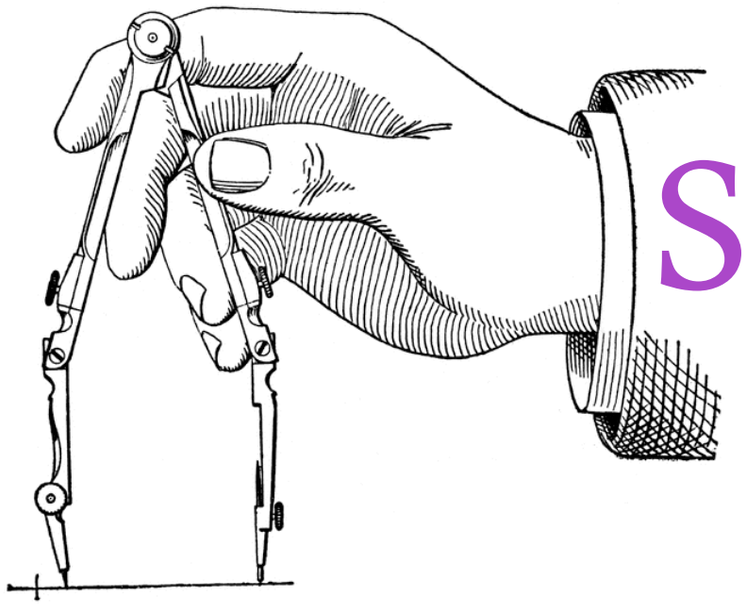
Richard E Sarkis
CSC 161: The Art of Programming

Class Administrivia



Agenda

- To be able to understand and write Python statements to output information to the screen
- To assign values to variables
- To get numeric information entered from the keyboard
- To perform a counted loop



Software Development Process

Software Development Process

The process of creating a program is often broken down into stages according to the information that is produced in each phase.

Software Development Process

1. Analyze the Problem
2. Determine Specifications
3. Create a Design
4. Implement the Design
5. Test/Debug the Program
6. Maintain the Program

Software Development Process

- **Analyze the Problem**
  Figure out exactly the problem to be solved. Try to understand it as much as possible.

Software Development Process

- **Determine Specifications**
- Describe exactly what your program will do.
  - Don't worry about how the program will work, but what it will do.
  - Includes describing the inputs, outputs, and how they relate to one another.

Software Development Process

- **Create a Design**
  - Formulate the overall structure of the program.
  - This is where the how of the program gets worked out.
  - You choose or develop your own algorithm that meets the specifications.

Software Development Process

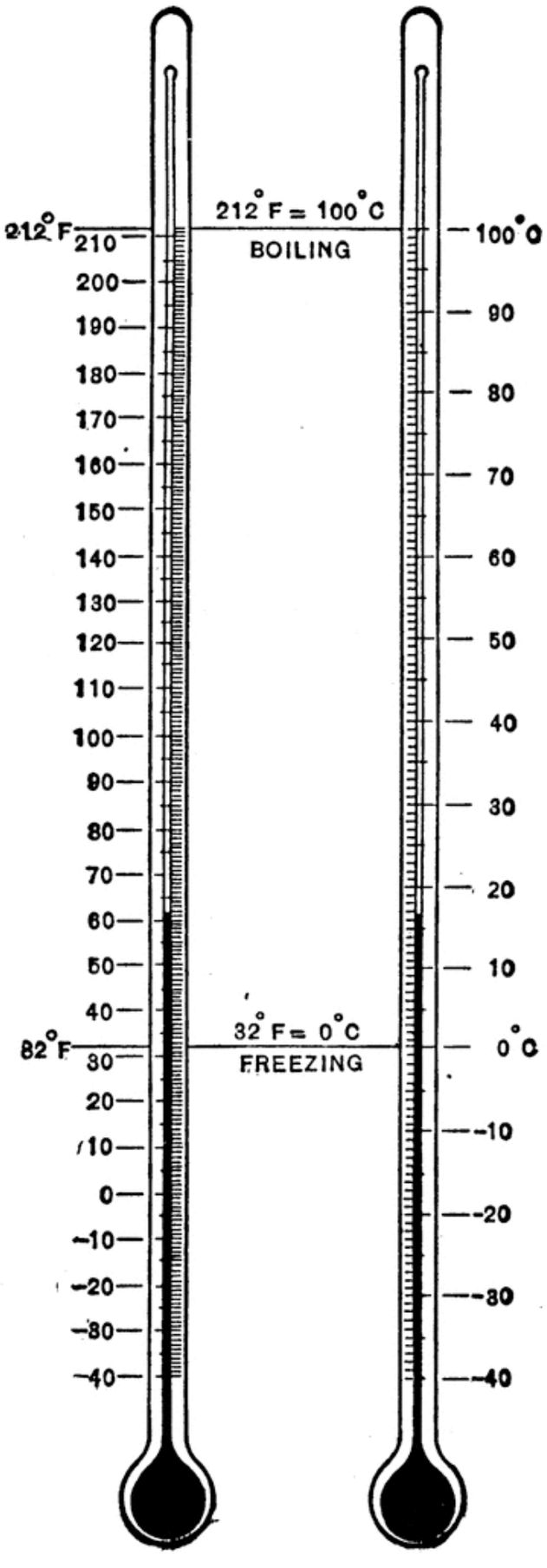
- **Implement the Design**
  - Translate the design into a computer language.
  - In this course we will use Python.

Software Development Process

- **Test/Debug the Program**
  - Try out your program to see if it worked.
  - If there are any errors (bugs), they need to be located and fixed. This process is called debugging.
  - Your goal is to find errors, so try everything that might "break" your program!

Software Development Process

- **Maintain the Program**
  - Continue developing the program in response to the needs of your users.
  - In the real world, most programs are never completely finished – they evolve over time.



Example Program: Temperature Converter

Example: Temperature Converter

1. **Analyze the Problem**
2. Determine Specifications
3. Create a Design
4. Implement the Design
5. Test/Debug the Program
6. Maintain the Program

- **Analysis**
- The temperature is given in *Celsius*, user wants it expressed in degrees *Fahrenheit*.

Example: Temperature Converter

1. Analyze the Problem
2. **Determine Specifications**
3. Create a Design
4. Implement the Design
5. Test/Debug the Program
6. Maintain the Program

- **Specification**
- Input: temperature in Celsius
- Output: temperature in **`Fahrenheit`**

$$T_{out}\text{°F} = 9/5(T_{in}\text{°C}) + 32$$

Example: Temperature Converter

1. Analyze the Problem
2. Determine Specifications
3. **Create a Design**
4. Implement the Design
5. Test/Debug the Program
6. Maintain the Program

- **Design**
  - Before we start coding, let's write a rough draft of the program in pseudocode
  - Pseudocode is precise English that describes what a program does, step by step.
  - Using pseudocode, we can concentrate on the algorithm rather than the programming language.

Example: Temperature Converter

1. Analyze the Problem
2. Determine Specifications
3. **Create a Design**
4. Implement the Design
5. Test/Debug the Program
6. Maintain the Program

- **Design: *Pseudocode***
  - Input the temperature in degrees Celsius
  - Calculate fahrenheit as (9/5)*celsius+32
  - Output fahrenheit
- Now we need to convert this to Python!

Example: Temperature Converter

1. Analyze the Problem
2. Determine Specifications
3. Create a Design
4. **Implement the Design**
5. Test/Debug the Program
6. Maintain the Program

```
# convert.py
# A program to convert Celsius temps to Fahrenheit
# by: Susan Computewell

def main():
    celsius = eval(input("What is the Celsius temperature? "))
    fahrenheit = (9/5) * celsius + 32
    print("The temperature is", fahrenheit,
          "degrees Fahrenheit.")

main()
```

Example: Temperature Converter

1. Analyze the Problem
2. Determine Specifications
3. Create a Design
4. Implement the Design
5. **Test/Debug the Program**
6. Maintain the Program

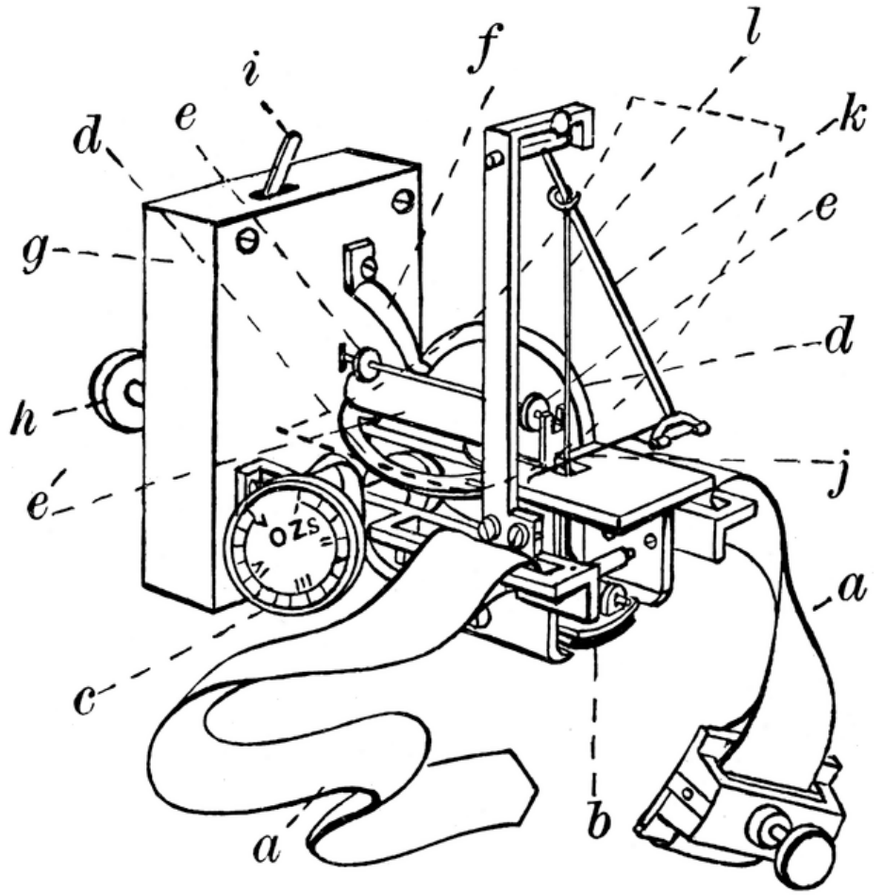
- **Implement**
  - Once we write a program, we should test it!

```
>>>
What is the Celsius temperature? 0
The temperature is  32.0  degrees Fahrenheit.
>>> main()
What is the Celsius temperature? 100
The temperature is  212.0  degrees Fahrenheit.
>>> main()
What is the Celsius temperature? -40
The temperature is  -40.0  degrees Fahrenheit.
>>>
```

Example: Temperature Converter

1. Analyze the Problem
2. Determine Specifications
3. Create a Design
4. Implement the Design
5. Test/Debug the Program
6. **Maintain the Program**

- **Maintain**
  - Keep your code updated!
  - Come back after, say, a year and review your code
  - If others use it, you'll support it for life!



Elements of Programs

Elements of Programs
Identifiers

- Names are given to data values (**`celsius`**, **`fahrenheit`**), modules (**`convert`**), functions (**`main`**), etc.
- These names are called **identifiers**, and they're used to describe objects in Python

Elements of Programs
Identifiers

- Required: identifiers must begin with a letter or underscore ("_"), followed by any sequence of letters, digits, or underscores
- Identifiers are case sensitive, e.g.

```
"FooBar" != "foobar"
```

Elements of Programs

Identifiers

- **These are all different, valid names:**

```
X
Celsius
Spam
spam
spAm
Spam_and_Eggs
Spam_And_Eggs
```

Elements of Programs
Identifiers

- **`NameError`** is the error when you try to use an identifier without a value assigned to it

```
>>> x = 5
>>> x
5
>>> print(x)
5
>>> print(spam)

Traceback (most recent call last):
  File "<pyshell#15>", line 1, in -toplevel-
    print spam
NameError: name 'spam' is not defined
>>>
```

Elements of Programs
Reserved Words

- Some words are part of Python itself
  - These words are known as *reserved words*
  - This means they are **not** available for you to use as a name for a variable, etc. in your program, e.g.

    `and`, **`del`**, **`for`**, **`is`**, **`raise`**, **`assert`**, **`elif`**, **`in`**
  - For a complete list: https://docs.python.org/3/reference/lexical_analysis.html#keywords

Elements of Programs

Literals

- Literals are values in source code that are written exactly as they are meant to be interpreted.
  - Strings, integers, floating point numbers, boolean values, etc.
  - `3.9`, `24`, `True`, `“Cheezburger”`

Elements of Programs
Expressions

- The fragments of code that produce or calculate new data value(s) are called expressions
- Simple identifiers *can also be expressions*
- They can consist of function calls, variable names, literals, and operators, etc.

Elements of Programs
Expressions

- Simpler expressions can be combined using operators.
  - `+`, `-`, `*`, `/`, `**`
- Spaces are irrelevant within an expression.
- The normal mathematical precedence applies.

```
((x1 – x2) / 2*n) + (spam / k**3)
```

Elements of Programs

Expressions

```
3 + 5
x
y == 22
```

Elements of Programs

Statements

- Statements are lines of code that 'do something'
- Expressions are often parts of statements

```
print(42)
if x == 22: do_y()
return
a = 7
```

Elements of Programs
Statements

- Output Statements
  - A **`print`** statement can print any number of expressions
  - Successive print statements will display on separate lines
  - A bare print will print a blank line

Elements of Programs

```
>>> print(3+4)
7
>>> print(3, 4, 3+4)
3 4 7
>>> print()

>>> print("The answer is", 3+4)
The answer is 7
>>>
```

Assignment Statements

Assignment Statements

- Simple Assignment

  `<variable> = <expr>`

  - **`variable`** is an identifier, **`expr`** is an expression
  - The expression on the *RHS* is evaluated to produce a value which is then associated with the variable named on the *LHS*

Assignment Statements

```
>>> x = 3.9 * x * (1-x)
>>> T_out  = 9/5 * (T_in) + 32
>>> x = 5
```

Assignment Statements

- Variables can be reassigned as many times as you want!

```
>>> myVar = 0
>>> myVar
0
>>> myVar = 7
>>> myVar
7
>>> myVar = myVar + 1
>>> myVar
8
>>>
```

Assignment Statements

- Variables are like a box we can put values in
- When a variable changes, the old value is erased and a new one is written in

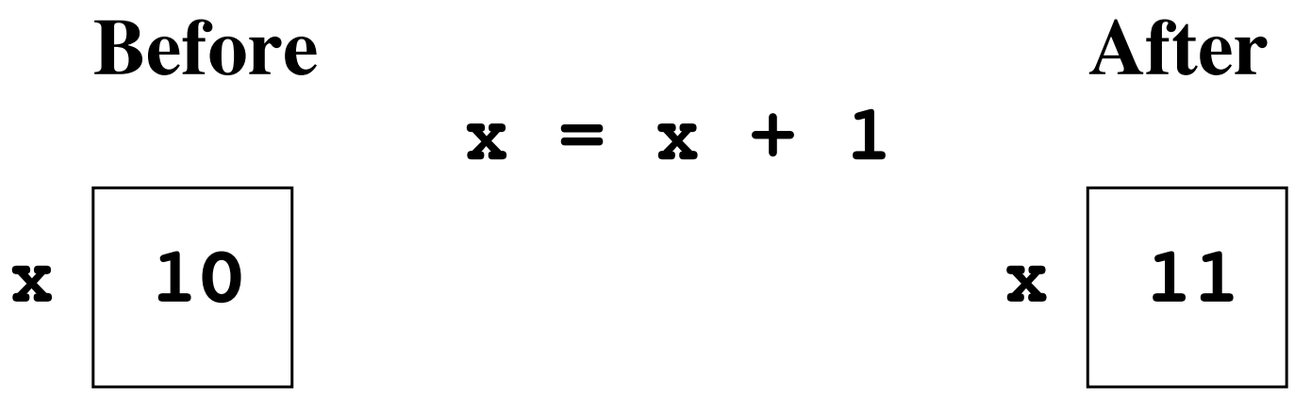


Figure 2.1: Variable as box view of `x = x + 1`

Assignment Statements

- However! Python doesn't overwrite these memory locations (boxes)
- Assigning a variable is more like putting a "sticky note" on a value and saying, "this is x"

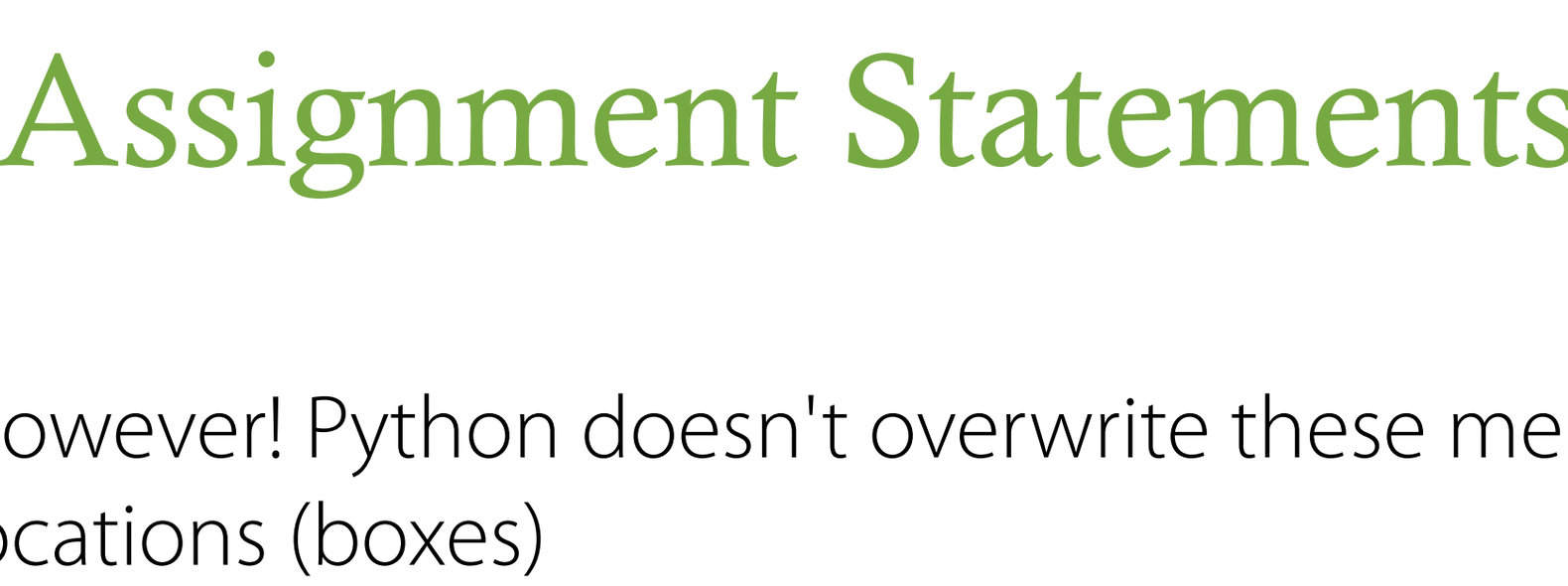
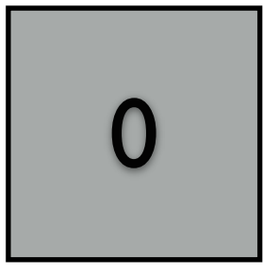
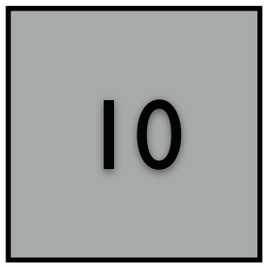


Figure 2.2: Variable as sticky note (Python) view of `x = x + 1`

In the C Language

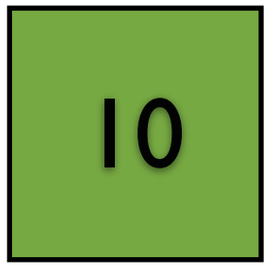
```
int num = 0;
```





In the Python Language

```
num = 0
```



Assignment Statements

User Input

- The purpose of an input statement is to get input from the user and store it into a variable

```
<variable> = eval(input(<prompt>))
```

Assignment Statements
User Input

- First the prompt is printed
- The *input* part waits for the user to enter a value and press <*enter*>
- The expression that was entered is evaluated to turn it from a string of characters into a Python value (a number)
- The value is assigned to the variable

Assignment Statements
Simultaneous Assignment

- Several values can be calculated at the same time

  `<var>, <var>, … = <expr>, <expr>, …`

- Evaluate the expressions in the RHS and assign them to the variables on the LHS

- `sum, diff = x+y, x-y`

Assignment Statements
Simultaneous Assignment

- How could you use this to swap the values for **`x`** and **`y`**?
- Why doesn't this work?

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

- We could use a temporary variable…

Assignment Statements
Simultaneous Assignment

- We can swap the values of two variables quite easily in Python!

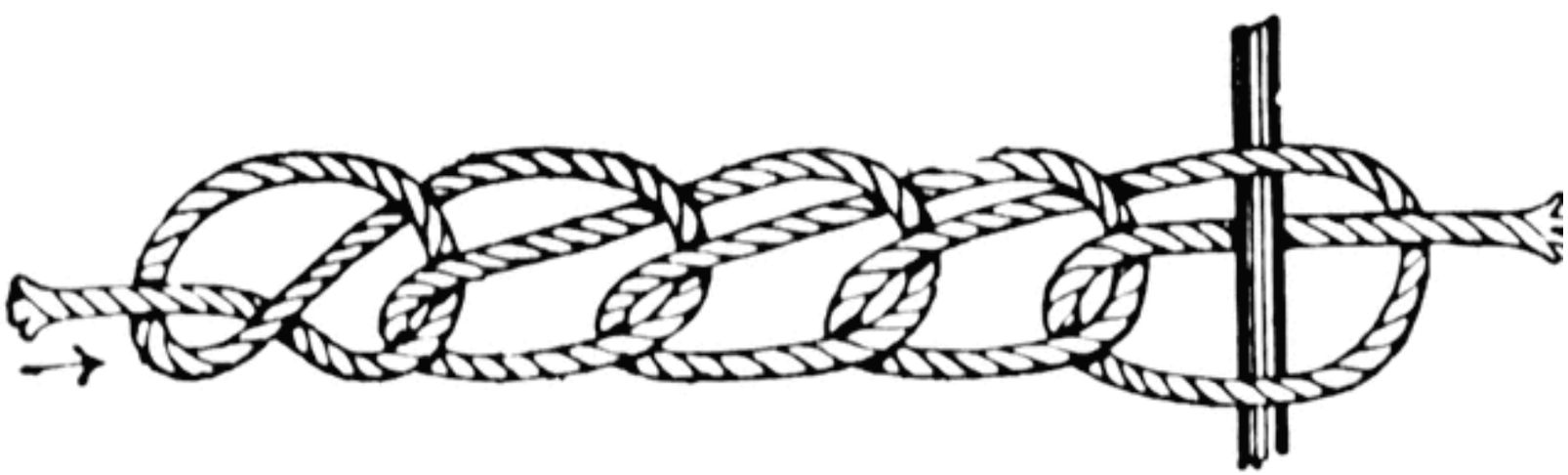
```
>>> x = 3
>>> y = 4
>>> print(x, y)
3 4
>>> x, y = y, x
>>> print(x, y)
4 3
```

Assignment Statements
Simultaneous Assignment

- We can use this same idea to assign multiple variables from a single input statement!
- Use commas to separate the inputs:

```
def spamneggs():
    spam, eggs = eval(input("Enter # of slices of spam followed by # of eggs: "))
    print ("You ordered", eggs, "eggs and", spam, "slices of spam. Yum!“)

>>> spamneggs()
Enter the number of slices of spam followed by the number of eggs: 3, 2
You ordered 2 eggs and 3 slices of spam. Yum!
>>>
```



Definite Loops

Definite Loops

- A definite loop executes a **definite** number of times, i.e., at the time Python starts the loop it knows exactly how many iterations to do.
- `for <var> in <sequence>:`
  `<body>`
- The beginning and end of the body are indicated by indentation

Definite Loops

- `for <var> in <sequence>:`
  `<body>`

  - The variable **`<var>`** after the **`for`** is called the *loop index*

  - It is assigned each successive value in **`<sequence>`**

Definite Loops

```
>>> for i in [0,1,2,3]:
        print (i)

0
1
2
3
>>> for odd in [1, 3, 5, 7]:
        print(odd*odd)

1
9
25
49

>>>
```

Definite Loops

- In **`chaos.py`**, what did **`range(10)`** do?

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- **`range`** is a built-in Python function that generates a sequence of numbers, starting with 0
- **`list`** is a built-in Python function that turns the sequence into an explicit list
- The body of the loop executes 10 times

Definite Loops

- `for` loops alter the flow of program execution, so they are referred to as **control structures**

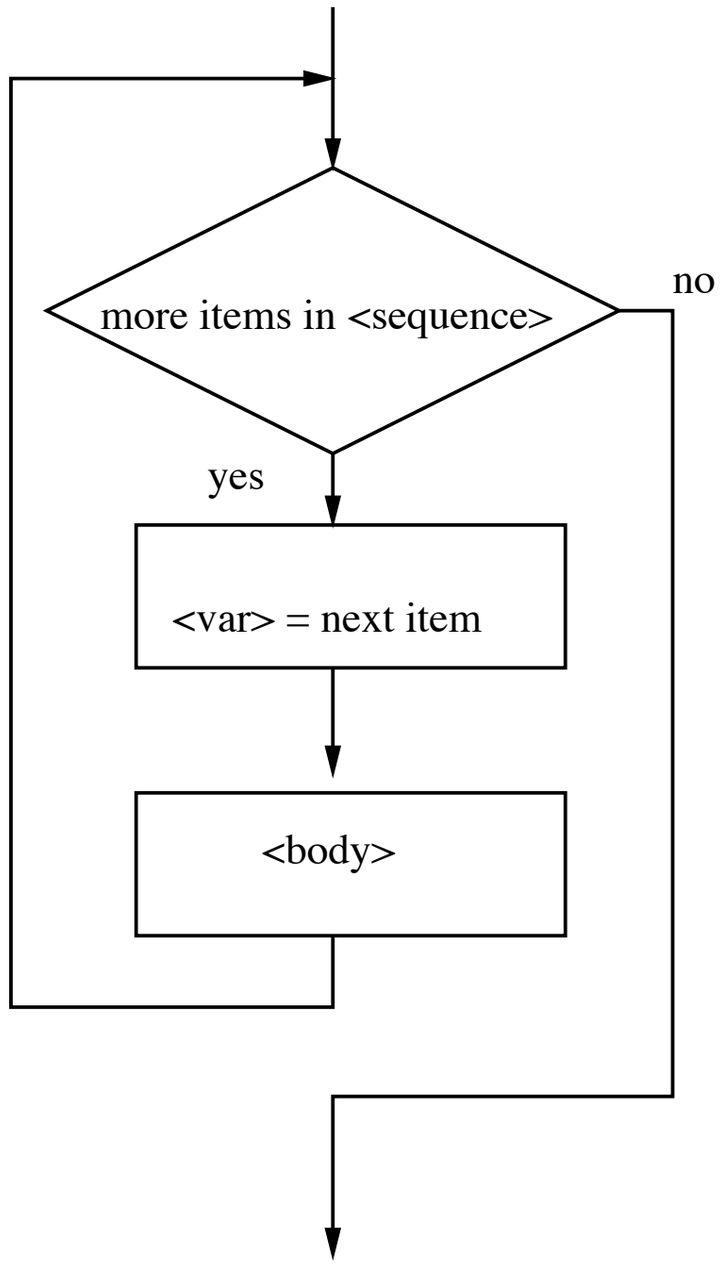
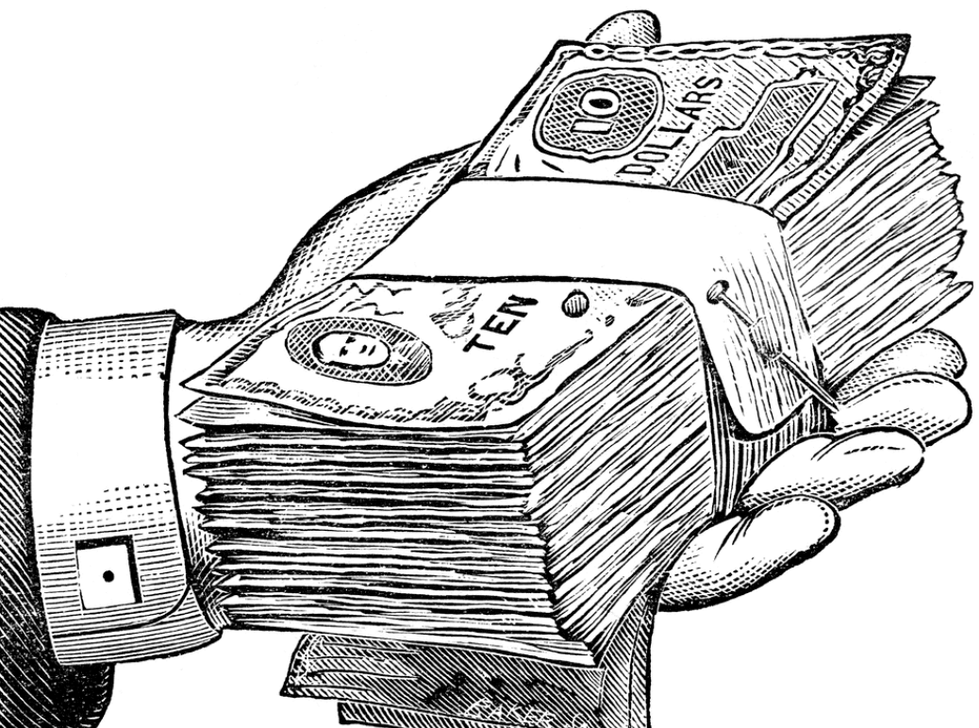


Figure 2.3: Flowchart of a `for` loop.

Example Program:

Future Value



Example: Future Value

1. **Analyze the Problem**
2. Determine Specifications
3. Create a Design
4. Implement the Design
5. Test/Debug the Program
6. Maintain the Program

- **Analysis**
  - Money deposited in a bank account earns interest
  - How much will the account be worth 10 years from now?
  - Inputs: principal, interest rate
  - Output: value of the investment in 10 years

Example: Future Value

1. Analyze the Problem
2. **Determine Specifications**
3. Create a Design
4. Implement the Design
5. Test/Debug the Program
6. Maintain the Program

- **Specifications**
  - User enters the initial amount to invest, the principal
  - User enters an annual percentage rate, the interest
  - The specifications can be represented like this…

Example: Future Value

1. Analyze the Problem
2. **Determine Specifications**
3. Create a Design
4. Implement the Design
5. Test/Debug the Program
6. Maintain the Program

- **Program** Future Value
- **Inputs:**
  principal The amount of money being invested, in dollars
  apr The annual percentage rate expressed as a decimal number.
- **Output**: The value of the investment 10 years in the future
- **Relationship:** Value after one year is given by `principal * (1 + apr)`
  - This needs to be done 10 times.

Example: Future Value

1. Analyze the Problem
2. Determine Specifications
3. **Create a Design**
4. Implement the Design
5. Test/Debug the Program
6. Maintain the Program

- **Design**
  - Print an introduction
  - Input the amount of the principal (`principal`)
  - Input the annual percentage rate (`apr`)
  - Repeat 10 times:
    - `principal = principal * (1 + apr)`
  - Output the value of `principal`

Example: Future Value

1. Analyze the Problem
2. Determine Specifications
3. Create a Design
4. **Implement the Design**
5. Test/Debug the Program
6. Maintain the Program

- **Implementation**
  - Each line translates to one line of Python (in this case)
  - Print an introduction

```
print ("This program calculates the future")
print ("value of a 10-year investment.")
```

  - Input the amount of the principal

```
principal = eval(input("Enter the initial
principal: "))
```

Example: Future Value

1. Analyze the Problem
2. Determine Specifications
3. Create a Design
4. **Implement the Design**
5. Test/Debug the Program
6. Maintain the Program

- **Implementation**
  - Input the annual percentage rate

```
apr = eval(input("Enter the annual interest rate: "))
```

  - Repeat 10 times:

```
for i in range(10):
```

  - Calculate principal

```
principal = principal * (1 + apr)
```

  - Output the value of the principal at the end of 10 years

```
print ("The value in 10 years is:", principal)
```

Example: Future Value

1. Analyze the Problem
2. Determine Specifications
3. Create a Design
4. **Implement the Design**
5. Test/Debug the Program
6. Maintain the Program

```
# futval.py
#    A program to compute the value of an investment
#    carried 10 years into the future

def main():
    print("This program calculates the future value of a 10-
year investment.")

    principal = eval(input("Enter the initial principal: "))
    apr = eval(input("Enter the annual interest rate: "))

    for i in range(10):
        principal = principal * (1 + apr)

    print ("The value in 10 years is:", principal)

main()
```

Example: Future Value

1. Analyze the Problem
2. Determine Specifications
3. Create a Design
4. Implement the Design
5. **Test/Debug the Program**
6. Maintain the Program

```
>>> main()
This program calculates the future value of a 10-year investment.
Enter the initial principal: 100
Enter the annual interest rate: .03
The value in 10 years is: 134.391637934
>>> main()
This program calculates the future value of a 10-year investment.
Enter the initial principal: 100
Enter the annual interest rate: .10
The value in 10 years is: 259.37424601
```

Coding in Style

PEP 8

Questions?