Loops In Python

In this section of notes you will learn how to rerun parts of your program without having to duplicate the code.

Application Of Loops In Actual Software

Re-running the entire program

```
Enter your age (must be non-negative):  -1
Enter your age (must be non-negative):  27
Enter your gender (m/f):
```

Re-running specific parts of the program

Basic Structure Of Loops

Whether or not a part of a program repeats is determined by a loop control (typically just a variable).

- Initialize the control to the starting value
- Testing the control against a stopping condition (Boolean expression)
- Executing the body of the loop (the part to be repeated)
- Update the value of the control

Types Of Loops

1.Pre-test loops

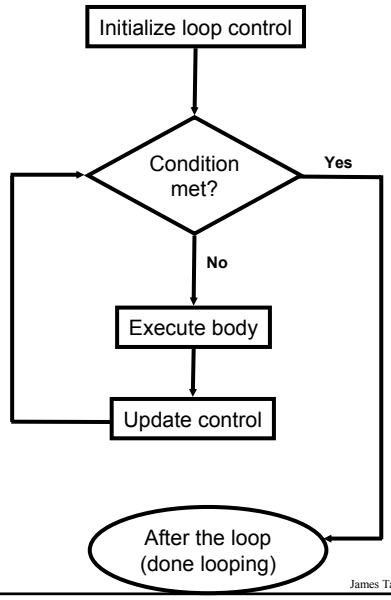
- Check the stopping condition *before* executing the body of the loop.
- The loop executes *zero or more* times.

2.Post-test loops

- Checking the stopping condition *after* executing the body of the loop.
- The loop executes *one or more* times.

Pre-Test Loops

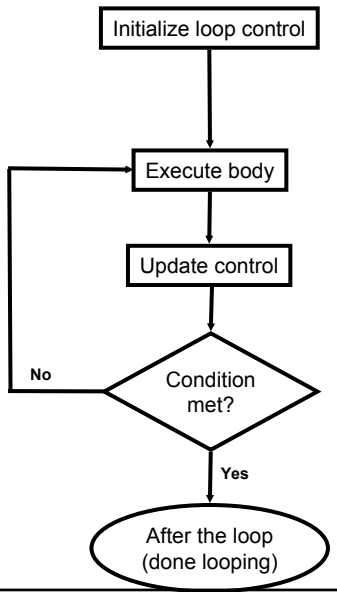
1. Initialize loop control
2. Check if the stopping condition has been met
   a. If it's been met then the loop ends
   b. If it hasn't been met then proceed to the next step
3. Execute the body of the loop (the part to be repeated)
4. Update the loop control
5. Go to step 2



James Tam

Post-Test Loops (Not Implemented In Python)

1. Initialize loop control (sometimes not needed because initialization occurs when the control is updated)
2. Execute the body of the loop (the part to be repeated)
3. Update the loop control
4. Check if the stopping condition has been met
   a. If it's been met then the loop ends
   b. If it hasn't been met then return to step 2.



James Tam

Pre-Test Loops In Python

1. While
2. For

Characteristics:

1. The stopping condition is checked *before* the body executes.
2. These types of loops execute zero or more times.

Post-Loops In Python

- Note: this type of looping construct has not been implemented with this language.
- But many other languages do implement post test loops.

Characteristics:

- The stopping condition is checked *after* the body executes.
- These types of loops execute one or more times.

The While Loop

- This type of loop can be used if it's *not known* in advance how many times that the loop will repeat (most powerful type of loop, any other type of loop can be simulated with a while loop).
- **Format:**
- (Simple condition)

```
while (Boolean expression):
    body
```

(Compound condition)

```
while (Boolean expression) Boolean operator (Boolean expression):
    body
```

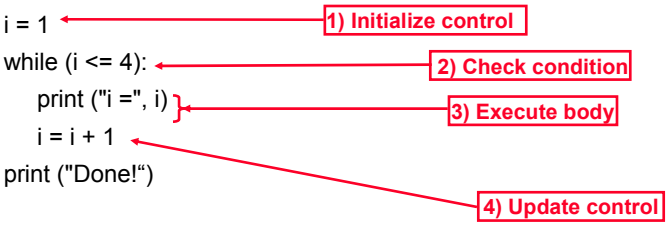

The While Loop (2)

- **Program name:** while1.py

```
i = 1
while (i <= 4):
    print ("i =", i)
    i += 1
print ("Done!")
```

The While Loop (2)

- **Program name:** while1.py



Tracing The While Loop

Execution	Variable
>python while1.py	i

The For Loop

- Typically used when it *is known* in advance how many times that the loop will execute (counting loop).
- **Syntax:**

  for <*name of loop control*> in <*something that can be iterated*>:

  body
- **Program name:** for1.py

```
total = 0;
for i in range (1, 5, 1):
  total = total + i
  print ("i=", i, " total=", total)
print ("Done!“)
```

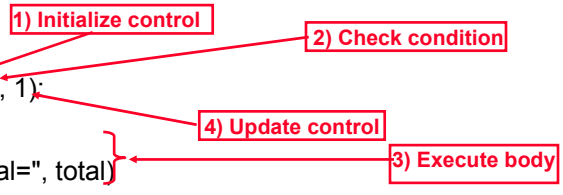

The For Loop

- Typically used when it *is known* in advance how many times that the loop will execute (counting loop).
- **Syntax:**

  for <*name of loop control*> in <*something that can be iterated*>:

  body
- **Program name:** for1.py

```
total = 0;
for i in range (1, 5, 1):
  total = total + i
  print ("i=", i, " total=", total)
print ("Done!“)
```



Tracing The First For Loop Example

Execution	**Variables**	
>python for1.py	i	total

Counting Down With A For Loop

- **Program name:** for2.py

```
for i in range (5, 0, -1):
    total = total + i
    print ("i = ", i, "\t total = ", total)
print ("Done!“)
```

Tracing The Second For Loop Example

Execution	Variables	
>python for2.py	i	total

Erroneous For Loops

- The logic of the loop is such that the end condition has already been reached with the start condition.
- **Example:**

```
for i in range (5, 0, 1):
        total = total + i
        print ("i = ", i, "\t total = ", total)
print ("Done!“)
```

Loop Increments Need Not Be Limited To One

•**While**

```
i = 0
while (i <= 100):
    print ("i =", i)
    i = i + 5
print ("Done!“)
```

•**For**

```
for i in range (0, 105, 5):
    print ("i =", i)
print ("Done!“)
```


Sentinel Controlled Loops

•The stopping condition for the loop occurs when the ‘sentinel’ value is reached.

•**Program name**: sum.py

```
total = 0
temp = 0
while (temp >= 0):
    temp = input ("Enter a non-negative integer (negative to end
 series):")
    temp = int(tem)
    if (temp >= 0):
        total = total + temp

print ("Sum total of the series:", total)
```

Sentinel Controlled Loops (2)

- Sentinel controlled loops are frequently used in conjunction with the error checking of input.
- Example:

```
selection = " "
while selection not in ("a", "A", "r",  "R", "m", "M", "q", "Q"):
    print "Menu options"
    print "(a)dd a new player to the game"
    print "(r)emove a player from the game"
    print "(m)odify player"
    print "(q)uit game"
    selection = raw_input ("Enter your selection: ")
    if selection not in ("a", "A", "r",  "R", "m", "M", "q", "Q"):
        print "Please enter one of 'a', 'r', 'm' or 'q'"
```


Recap: What Looping Constructs Are Available In Python/When To Use Them

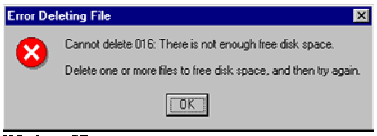
Construct	When To Use
Pre-test loops	You want the stopping condition to be checked before the loop body is executed (typically used when you want a loop to execute zero or more times).
• While	• The most powerful looping construct: you can write a 'while-do' loop to mimic the behavior of any other type of loop. In general it should be used when you want a pre-test loop which can be used for most any arbitrary stopping condition e.g., execute the loop as long as the user doesn't enter a negative number.
• For	• A 'counting loop': You want a simple loop to repeat a certain number of times.
Post-test: None in Python	You want to execute the body of the loop before checking the stopping condition (typically used to ensure that the body of the loop will execute at least once). The logic can be simulated in Python however.

User-Friendly Software

- In today's world it's not just sufficient to create software that has implemented a set of operations.
- If the person using the system cannot understand it or has troubles using common functions then the software or technology is useless.
- Reference course: If you're interested in more information:
  - http://pages.cpsc.ucalgary.ca/~tamj/2008/481W/index.html

Not So Friendly Examples

Do I have any choice in this? [AXE a hex editor]



Windows 95

Uhhh... I give up on this one [Mac shareware version of RISK]

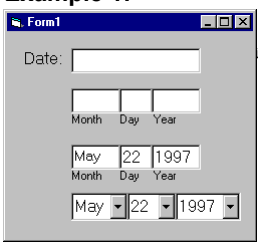
Some Rules For Designing Software

- (The following list comes from Jakob Nielsen's 10 usability heuristics from the book "*Usability Engineering*"
  1. Minimize the user's memory load
  2. Be consistent
  3. Provide feedback
  4. Provide clearly marked exits
  5. Deal with errors in a helpful and positive manner

1. Minimize The User's Memory Load

- Describe required the input format, use examples, provide default inputs
- Examples:

Example 1:



Example 2:

```
[csc loops 25 ]> python hci.py
Enter your birthday <month> <day> <year> e.g., 11 17 1977
Birthday:
```

2. Be Consistent

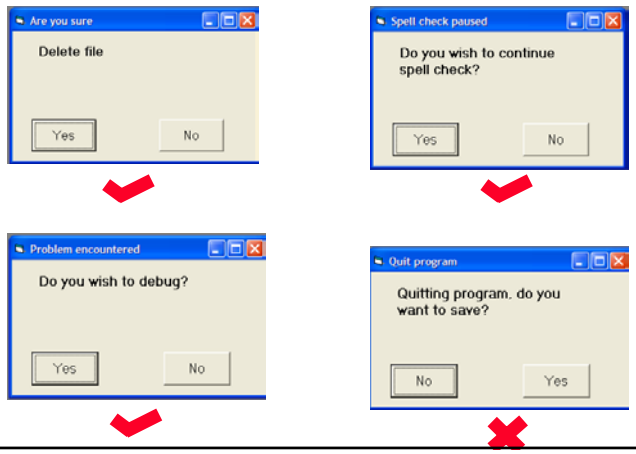
•Consistency of effects

- Same words, commands, actions will always have the same effect in equivalent situations
- Makes the system more predictable
- Reduces memory load

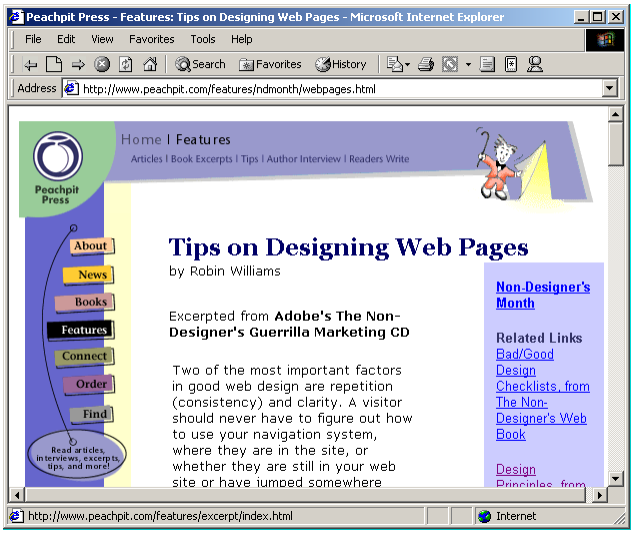
2. Be Consistent

•Consistency of language and graphics

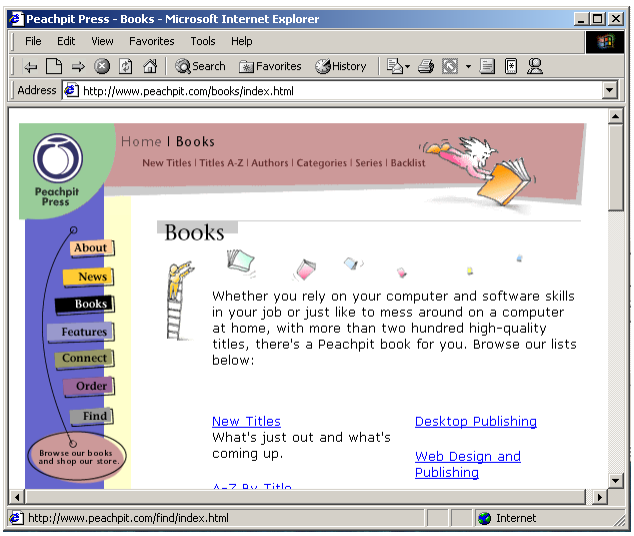
- Same information/controls in same location on all screens / dialog boxes forms follow boiler plate.
- Same visual appearance across the system (e.g. widgets).



2. Be Consistent



2. Be Consistent



2. Be Consistent

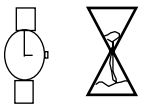
This last option allows the user to proceed to the next question.

```
FIRST CATEGORY: ELECTRICITY
---------------------------
You can either enter your monthly kilowatt hours or have an estimate
based on the size of the accomodation that you live in.

(e)stimate
(k)ilowatt hours used
(q)uit this question and proceed to the next question
Enter selection: q

Tons of carbon generated from powering accomodation: 0
Current tons of carbon currently generated: 0


SECOND CATEGORY: HEATING
------------------------

What size of place do you live:
(s)mall house or a flat
(m)edium house
(l)arge house
(q)uit this question and proceed to the next question
Enter selection:
```

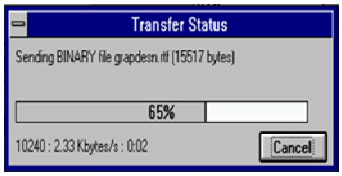
James Tam

3. Provide Feedback

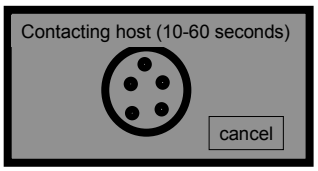
- What is the program doing?



Cursor



Progress bar

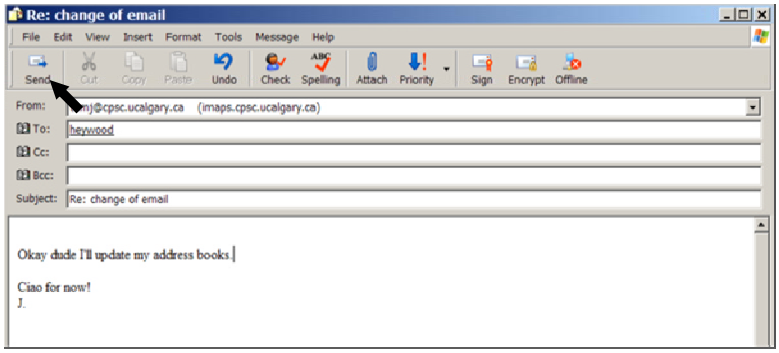


Random graphic

James Tam

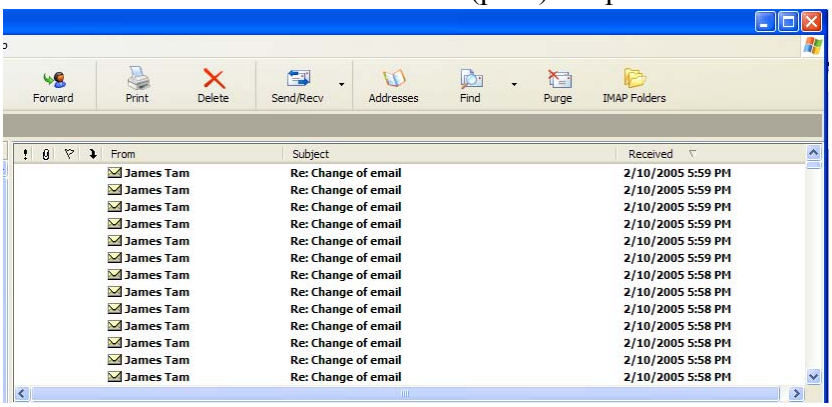
3. Provide Feedback

•What is the program doing?



3. Provide Feedback

•The rather unfortunate effect on the (poor) recipient.



3. Provide Feedback

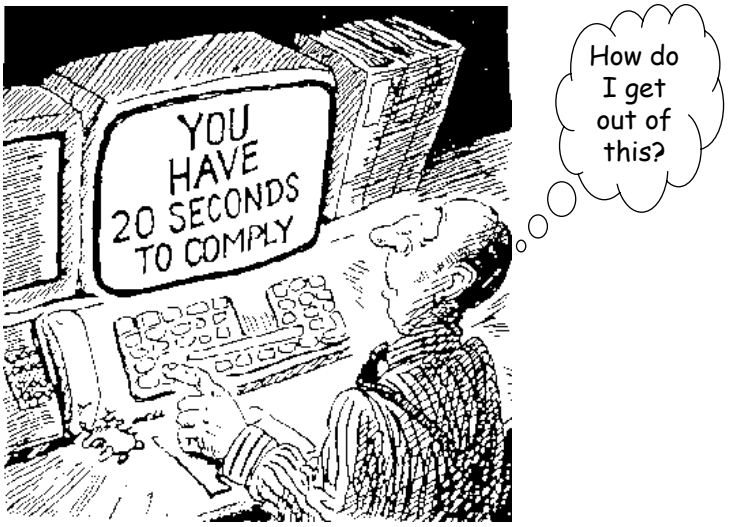
- In terms of this course, letting the user know:
  - what the program is doing (e.g., opening a file),
  - what errors may have occurred (e.g., could not open file),
  - and why (e.g., file "input.txt" could not be found)

...is not hard to do and not only provides useful updates with the state of the program ("Is the program almost finished yet?") but also some clues as to how to avoid the error (e.g., make sure that the input file is in the specified directory).

- At this point your program should at least be able to provide some rudamentary feedback
  - E.g., if a negative value is entered for age then the program can remind the user what is a valid value (the valid value should likely be shown to the user as he or she enters the value):
  - age = input ("Enter age (0 – 114): ")

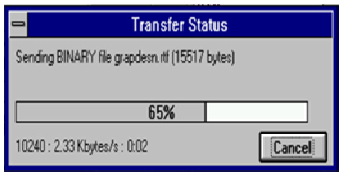
4. Provide Clearly Marked Exits

- User's should never feel 'trapped' by a program.



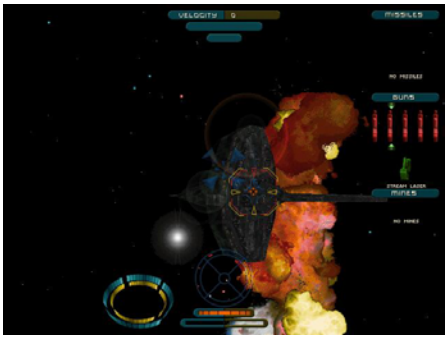
4. Provide Clearly Marked Exits

- This doesn't just mean providing an exit from the program but the ability to 'exit' (take back) the current action.
  - Universal Undo/Redo
    - e.g., <Ctrl>-<Z> and <Ctrl>-<Y>
  - Progress indicator & Interrupt
  - Length operations



4. Provide Clearly Marked Exits

- Restoring defaults
  - Getting back original settings



Wing Commander: Privateer 2 © Origin-EA

4. Provide Clearly Marked Exits

The user can skip any question

```
FIRST CATEGORY: ELECTRICITY
---------------------------
You can either enter your monthly kilowatt hours or have an estimate
based on the size of the accomodation that you live in.

(e)stimate
(k)ilowatt hours used
(q)uit this question and proceed to the next question
Enter selection: q

Tons of carbon generated from powering accomodation: 0
Current tons of carbon currently generated: 0


SECOND CATEGORY: HEATING
------------------------

What size of place do you live:
(s)mall house or a flat
(m)edium house
(l)arge house
(q)uit this question and proceed to the next question
Enter selection:
```


5. Deal With Errors In A Helpful And Positive Manner

Error handling in "The good 'ole days"

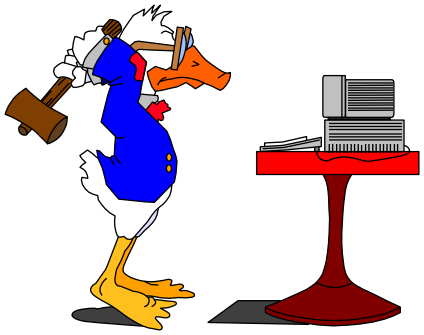
What is "error 15762"?

Rules Of Thumb For Error Messages

1. Polite and non-intimidating
   - Don't make people feel stupid
     – Try again, bonehead!
2. Understandable
   - Error 25
3. Specific
   - Cannot open this document
   - Cannot open "chapter 5" because the application "Microsoft Word" is not on your system
4. Helpful
   - Cannot open "chapter 5" because the application "Microsoft Word" is not on your system. Open it with "WordPad" instead?

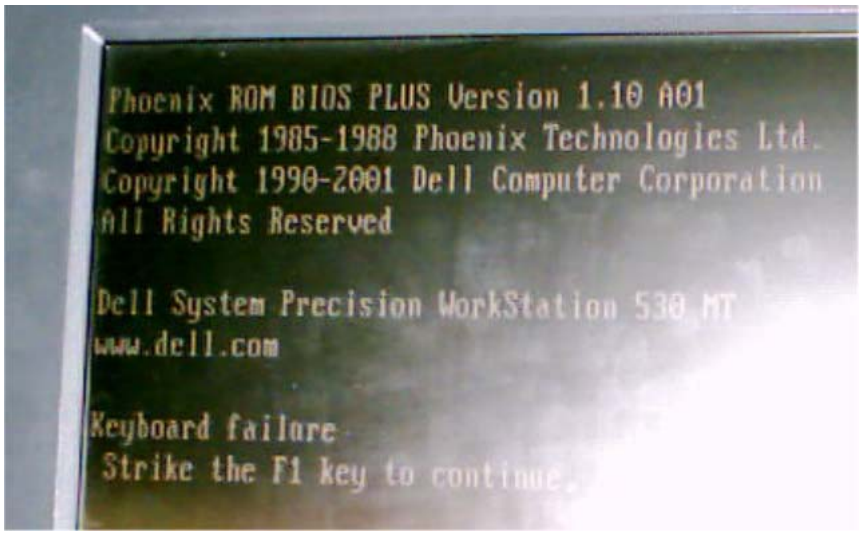
Examples Of Bad Error Messages

Microsoft's NT Operating System



“HIT ANY KEY TO CONTINUE”

I Think I'd Rather Deal With The Any Key!!!



Picture courtesy of James Tam: An error message from a Dell desktop computer

Solving A Problem Using Loops

- Problem: Write a program that will execute a game:
  - The program will randomly generate a number between one and ten.
  - The player will be prompted to enter their guess.
  - The program will continue the game until the player indicates that they no longer want to continue.

- **Program name**: guessingGame.py

Guessing Game

```
guess = 0
answer = 0
choice = "Y"
while choice not in ("q", "Q"):
    answer = random.randrange (10) + 1
    guess = int(input ("Enter your guess: "))
    if (guess == answer):
        print ("You guessed correctly!“)
    else:
        print ("You guessed incorrectly“)
    print ("Number was", answer, ", your guess was", guess)
    print ("Play again?  Enter 'q' to quit, anything else to play again“)
    choice = input("Choice: ")
    print ()
print ("Exiting game“)
```


Infinite Loops

- Infinite loops never end (the stopping condition is never met).
- They can be caused by logical errors:
  - The loop control is never updated (Example 1 – below).
  - The updating of the loop control never brings it closer to the stopping condition (Example 2 – next slide).
- **Example 1**: infinite1.py

```
i = 1
while (i <=10):
    print ("i = ", i)
i = i + 1
```

To stop a program with an infinite loop in Unix simultaneously press the <ctrl> and the <c> keys

Infinite Loops (2)

•**Example 2**: infinite2.py

```
i = 10
while (i > 0):
    print ("i = ",  i)
    i = i + 1
```

To stop a program with an infinite loop in Unix simultaneously press the <ctrl> and the <c> keys

Nested Loops

•One loop executes inside of another loop(s).
•Example structure:

```
Outer loop (runs n times)
    Inner loop (runs m times)
        Body of inner loop (runs n x m times)
```

• Program name: nested.py

```
for i in range (1, 3, 1):
    for j in range (1, 4, 1):
        print ("i = ", i, " j = ", j)
print ("Done!“)
```

Testing Loops

- Make sure that the loop executes the proper number of times.
- Test conditions:
  1) Loop does not run
  2) Loop runs exactly once
  3) Loop runs exactly ‘n’ times

Testing Loops: An Example

```
sum = 0
i = 1
last = 0

last = input ("Enter the last number in the sequence to sum : ")
while (i <= last):
   sum = sum + i
   print ("i = ", i)
   i = i + 1

print ("sum =", sum)
```

After This Section You Should Now Know

- When and why are loops used in computer programs
- What is the difference between pre-test loops and post-test loops
- How to trace the execution of pre-test loops
- How to properly write the code for a loop in a program
- Some rules of thumb for interaction design
  1. Minimize the user's memory load
  2. Be consistent
  3. Provide feedback
  4. Provide clearly marked exits
  5. Deal with errors in a helpful and positive manner
- What are nested loops and how do you trace their execution
- How to test loops