

# Control Statements

## Selection: if and if-else Statements

- Allows system to make choices based on conditions
- **if** statement:
  - One-way selection statement (simplest form of selection)
- **if-else** statement:
  - Two-way selection statement
  - Used to check input for errors
- **if, elif, else** statements:
  - Multi-way selection statement
  - Used when program may be faced with testing conditions that entail more than two alternative courses of action

## Loops

- Two types of loops:
  - Definite iteration: repeat action for predefined number of times
  - Indefinite iteration: perform action until program determines it needs to stop
- Repetition statements repeat an action
- Each repetition known as **pass** or **iteration**
- **for Loop:** used when exact number of times for execution is known
  - General form:

```
Initialise variables  
for i = x to y:  
    (process)  
endFor  
print results
```

- `range( )`:
  - ◆ Off-by-one Error:
    - ◆ Count through range of number: `list(range(3))` → 0, 1, 2 #default 0 lower bound
    - ◆ Specify explicit lower bound: `list(range(1,4))` → 1, 2, 3

- Specifying steps:
  - ◆ **range( )** expects a third argument that allows for specifying a **step value**
  - ◆ `list( range(1, 6, 1) )` → [1, 2, 3, 4, 5] #same as using two arguments
  - ◆ `list( range(1, 6, 2) )` → [1, 3, 5] #uses every other number
  - ◆ `list( range(1, 6, 3) )` → [1, 4] #uses every third number
  - ◆ `list( range(1, 6, -1) )` → [6, 5, 4, 3, 2] #counts down
- CAUTION:
  - ◆ Do not alter control variable: a statement in the **for** loop should never assign a value to the control variable

```

for i = 1 to 10:
    print i
    i = i + 2      # don't do this
endFor

```

- ◆ Clear indentation and indication of loop body is important: block of code containing **for** loop must end with **endFor**

- **while Loop:** used for conditional iteration
- General form:

```

while (test condition):
    (process)
endWhile

```

- Computer starts by testing the **while** condition.
- If it's true the entire loop body is executed.
- The control is returned to the top to retest the **while** condition.
- The process is repeated as long as the **while** condition is true.

- **repeat-until Loop:** used for conditional iteration
- General form:

```

repeat
    (process)
until (test condition)

```

- The entire loops is always executed a first time, then the **repeat-until** condition is tested.
- If condition is not true, the entire loop is executed again, and the condition is tested again.
- The process is repeated until condition is true.
- Data Sentinel-Controlled Loops: user terminate entry when he chooses by entering appropriate signal known as sentinel
- **Type A: Using y / n question**

*BEGIN*

*Initialise variables*

*repeat*

*(process)*

*print "enter y to continue, n to stop"*

*read answer*

*until (answer == 'n' or answer == 'N')*

*print results*

*END*

- **Type B: repeat-until Loop with Phony Value Data Sentinel**
  - ◆ Reduces amount of data entry by user
  - ◆ Eliminates use of y / n question
  - ◆ Uses phony value **-1** as sentinel

*BEGIN*

*Initialise variables*

*repeat*

*print "enter value or -1 to stop"*

*read answer*

*if answer != -1:*

*(process)*

*endif*

*until (answer == '-1')*

*print results*

*END*

- **Type C: while Loop with Phony Data Sentinel**
  - ◆ Similar to Type B that uses **-1** as sentinel

*BEGIN*

```
Initialise variables
print "enter value or -1 to stop"
read answer
while answer != -1:
    (process)
    print "enter value or -1 to stop"
    read answer
endWhile
print results
END
```