CCPS109
Computer Science I
L1

Lecturer: Nhan Tran

n3tran@Ryerson.ca

Agenda

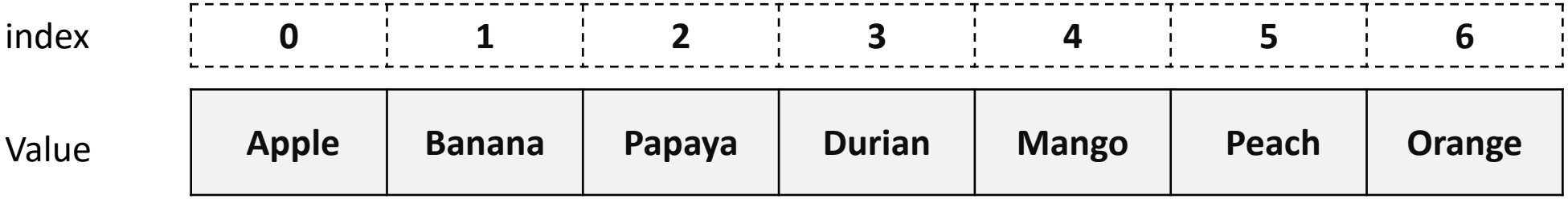
Announcement

Lecture:

list intro
iteration

List intro

- Simple explanation: list is abstraction items on list is indexable



fruit_list=['Apple','Banana','Papaya','Durian','Mango','Peach','Orange']

List intro

fruit_list=['Apple','Banana','Papaya','Durian','Mango','Peach','Orange’]

fruit_list[2] 'Papaya'

fruit_list[5] 'Peach'

List intro

list.append(x)

- Add an item to the end of the list. Equivalent to a[len(a):] = [x].

list.insert(i, x)

- Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

list.remove(x)

- Remove the first item from the list whose value is equal to x. It raises a ValueError if there is no such item.

List intro

list.pop([i])

- Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the i in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

list.clear()

- Remove all items from the list. Equivalent to del a[:].

list.count(x)

- Return the number of times x appears in the list.

Iteration (aka Loops)

- Doing the same thing many times
- Instead of writing out the same code again and again. We use iterations/ loops.
- Like conditionals: the conditional is evaluated then execute the code block(loop body).
- Once done with the loop body, recheck the condition to see if it still true.
- Jump to next code block if condition is false

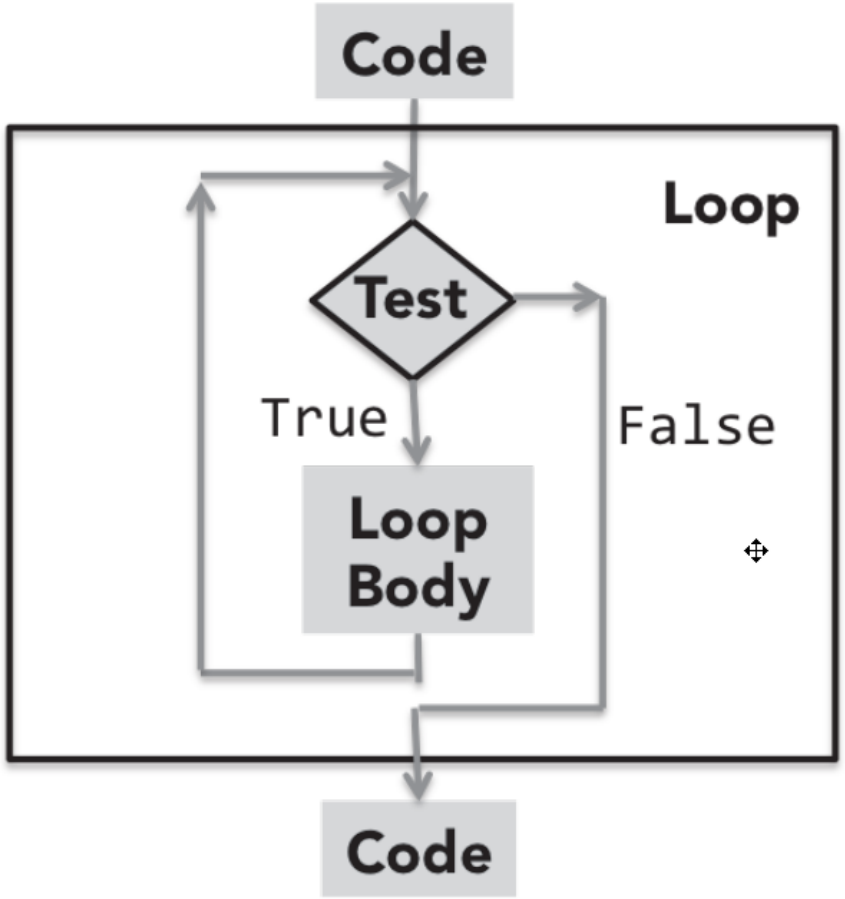


Figure 2.4 Flow chart for iteration

Iteration (aka Loops)

- While loop

```
i = 1
while i <= 12:
   print(i)
   i += 1   #i=i+1
Print(" I am outside of
while loop")
```

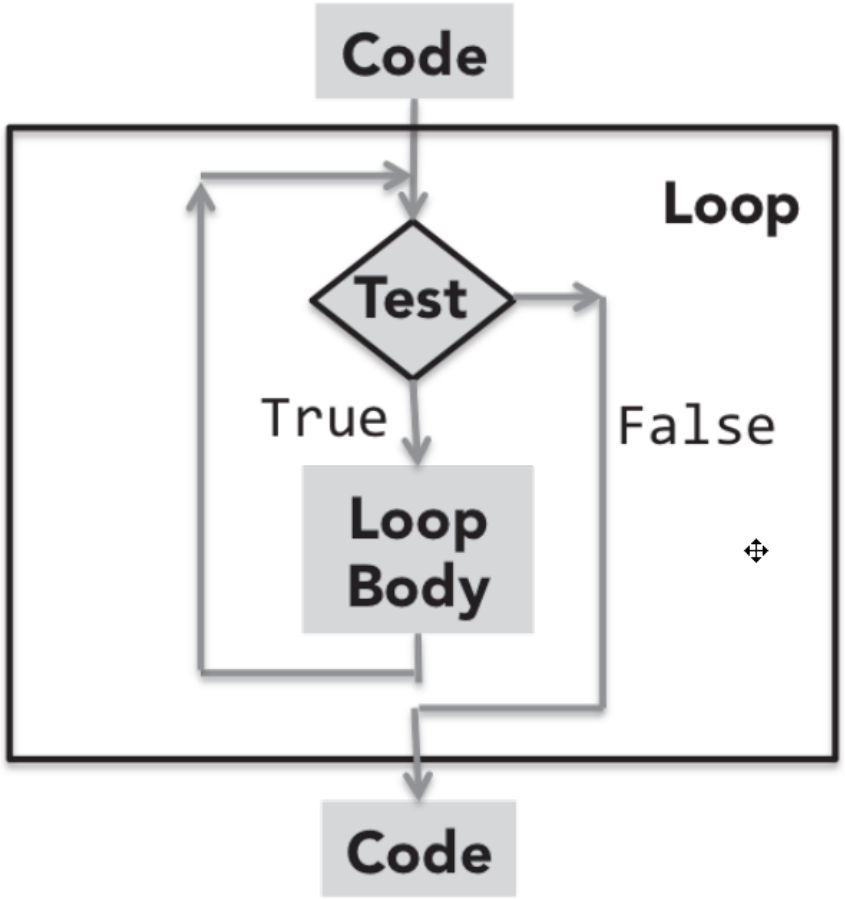


Figure 2.4 Flow chart for iteration

While

```
i = 1
while i <= 12:
   print(i)
   if i==5:
      break    #exit while loop
   i += 2
Print(“ I am outside of while loop”)
```

While

```
i = 1
while i <= 12:
   i += 2
   if i==5:
      continue    #skip the rest of loop code
                   #continue to the next iteration
   print(i)
Print(“ I am outside of while loop”)
```

Look up: while loop with else

While: recap

- `while <condition>:`
  `<expression>`
  `<expression>`
  `...`
- `<condition>` evaluates to True or False
- When `<condition>` is `True`, do all steps inside the while code block
- check `<condition>` again,
- repeat until `<condition>` is False, then execute code outside of block

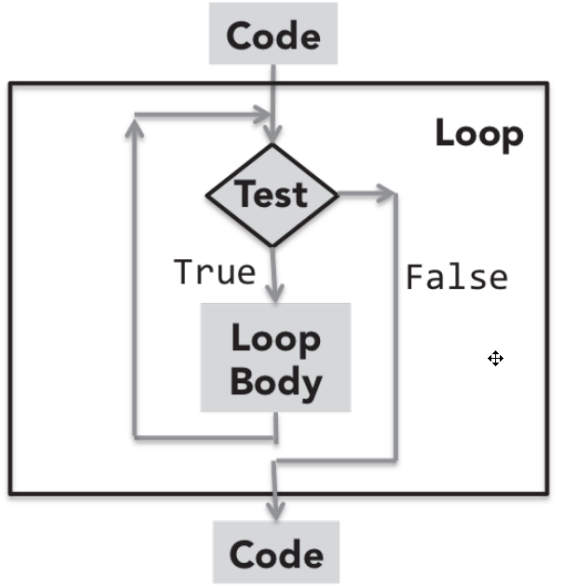


Figure 2.4 Flow chart for iteration

while and for loops

- iterate through numbers in a sequence

```
# while loop
n = 0
while n < 7:
   print(n)
   n = n+1
```

-
```
# simpler with for loop
for n in range(7):
   print(n)
```

For loop

```
for val_at_index in sequence:
    <expression>
    <expression>
    ...
```

- each time through the loop, `<val_at_index>` takes a value store at index
- first time, `< val_at_index >` starts at the lowest index
- next time, `< val_at_index >` gets the previous value + 1

For

```
#list of fruits name
fruits = ['Apple', 'Banana','Papaya','Durian','Mango','Peach','Organge']

for val in fruits:
    if val=='Durian':
        print('My favourite:'+val)
    else:
        print('meh:'+val)
```

For

```
fruits = ['Apple', 'Banana','Papaya','Durian','Mango','Peach','Organge']

for val in fruits:
    if val=='Durian':
        print('My favourite:'+val)
    else:
        print('meh:'+val)
```

Breaking loop

```
fruits = ['Apple', 'Banana','Papaya','Durian','Mango','Peach','Organge']

for x in fruits:
      if x == “Durian":
          break                    #exit for loop
      print(x)
```

loop ‘continue’

```
fruits = ['Apple', 'Banana','Papaya','Durian','Mango','Peach','Organge']
```

```
for x in fruits:
      if x == “Durian":
          continue
      print(x)
```

```
# stop the current iteration and
continue to the next.

#Will not print ‘Durian’
```

Loop

```
# List of numbers
numbr = [ 5, 3, 8, 4, 2, 6, 9, 11]

for val in numbr:
    sum = sum+val
print("The sum is", sum)

#hmm… something seem to be missing
```

Loop

```
# List of numbers
numbr = [ 5, 3, 8, 4, 2, 6, 9, 11]
sum=0
for val in numbr:
    sum = sum+val
print("The sum is", sum)
```

For loop

```
for <variable> in range(<some_num>):
    <expression>
    <expression>
    ...
```

- each time through the loop, `<variable>` takes a value
- first time, `<variable>` starts at the smallest value
- next time, `<variable>` gets the prev value + 1

Range

- default values are `start = 0` and `step = 1`
- loop until value is (`stop - 1`)

```
for n in range(7):              #0<=n<7  or 0<=n<=6
   print(n)
```

Range(start, stop, step)

- start value, stop value and optional step value
- Loop iterate until value is (stop – 1)

```
sum = 0
for i in range(7, 10):
    sum =sum + i
print('my sum:',sum)
```

#my sum: 24

```
#-----------------------
sum = 0
for i in range(5, 11, 2):
    sum += i
print("my sum:",sum)
```

#my sum: 21

Nested for

```
fruit_num=[3,4,2,24]
fruit_name=['Apple','Banana','Papaya','Durian','Mango','Peach','Orange’]

for x in fruit_num:
  for y in fruit_name:
    print(x*y)                  # try this=>print(x,y)
```

While:

- Unbounded number of iterations
- Can use for counter, but need initialization before loop
- Can end early via break
- ***May not*** be able to rewritable using `for` loop

For:

- Known number of iterations
- Can end early via break
- Use as counter
- rewritable using `while` loop

Coding so far

Basic constructs:

Numbers, assignments, input/output, comparisons, looping

Theoretically this is all you need , i.e. it is Turing complete:

“all problems can be solve via computation is solvable using only those you have already seen.”

However, the way we having writing program is ‘monolithic’

monolithic codes

- Easy for small-scale problems
- Hard to maintain for larger problems
  - Codes and details
- Reusability
- How to deal with separation of concerns

More code ≠ better programme

Black box

- Car
- TV
- Projector

- Abstraction of idea: No need to know how things work in order to used it.

Decomposition

Car: engine, drive train, gearbox, springs, computers, sensor, lights, breaks…

- All these components work to getting to allow use to drive safely

- Decomposition idea: different devices/components work together to achieve an end goal.

Decomposition for modularity

- In programming:

We separate code into **modules**

- **self-contained**
- **reusable**
- keep code **organized**
- **keep code coherent**
- decomposition can be achieve with **functions**

- Later on, decomposition via **classes**

Abstraction: need to know?

- A module of code can be consider as a black box
  - Cannot see internal detail
  - Nor should we need to see it
  - Hide coding details

  - Abstraction achieve via: documentation and function spefication

Function

- Reusable piece of code that carry out calculation for a piece of functionality
- Not run in program until invocation or called

Function:

- has a name
- **parameters** (0 or more)
- a **body**
- **returns** something
- **docstring** (optional but recommended)

Function

def name_of_function (list of paramenters):

Body of function

```
def printValue(x):
   print(x)

#invocation
printValue(34)
```

```
def findMax(x,y):
   val=0
   if x<y:
         val=y
   else:
        val=x
   return val

#invocation
max=findMax(55,12)
```

Function arguments

- Arguments are passed by **value**
- function parameters bind to passed values
- Variable and params belong to function are

local and accessible only inside function

- Unless declared otherwise

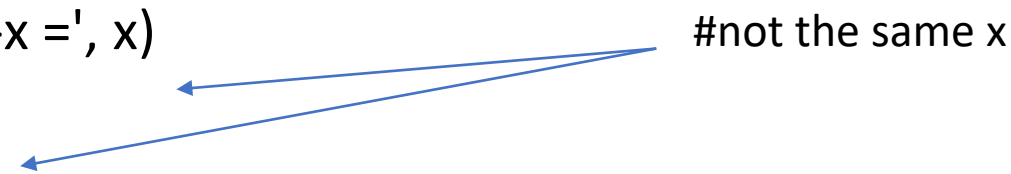
c=findMax(4,7)

print(c)

```
#logically the same
#as one in prior
#slide
def findMax(x,y):
  val=x
  print(x,y)
  if x<y:
        val=y
  return val
```

Scope

```
def f(x): #name x used as formal parameter
    y = 1
    x = x + y
    print('F-x =', x)
    return x
x = 3
y = 2
z = f(x) #value of x used as actual parameter
print('z =', z)
print('x =', x)
print('y =', y)
```



Function with unknow number of param

```
def my_function(*ffruits):
  print(“my 3rd fruit " + ffruits[2])

my_function('Apple',
'Banana','Papaya','Durian','Mango','Peach','Orga
nge')
```

Function with unknow number of param

```
def my_function(*ffruits):
    for fruit in ffruits:
        print('fruit item: '+fruit)

my_function('Apple',
'Banana','Papaya','Durian','Mango','Peach','Orga
nge')
```

End?