Lecture 4

CORE ELEMENTS
COLLECTIONS
&
REPETITION

What we have seen so Far

- What variables are
- Introduced the notion of Functions & Parameters
- Selection Structure (branching)
- How to do simple repetitions/iterations

Overview

- How to represent a collection of values?
  - Tuples
  - Lists
- Mutable, Immutable object
- For Loops
- While Loops

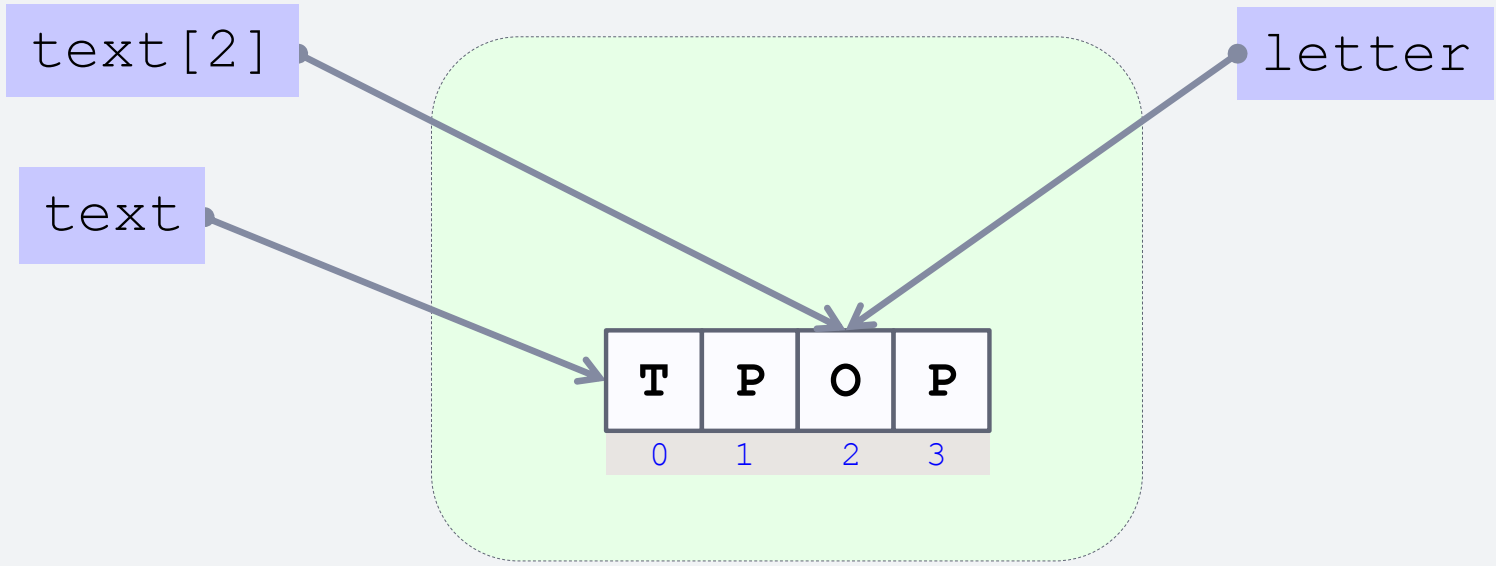
Data Types

- We have seen numbers
  - int: whole number
  - float: decimal point
  - boolean (True, False)
- String
  - str: “a word”
- What about a collection of data?

String (str)

- A kind of collection of characters

```
>>> text = 'TPOP'
>>> letter = text[2] # Accessing the third element
>>> letter
'O'
```



Tuples in Python

- Tuples are immutable objects, e.g. we CANNOT modify their contents.

Code

```
>>> my_tuple = (1, 10, 4, 5) # tuple creation ( ,)
>>> my_tuple[2]      # Accessing the third element
4
>>> my_tuple
(1, 10, 4, 5)
>>> my_tuple[2] = 9 # Modifying the third element

Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    my_tuple [2] = 9
TypeError: 'tuple' object does not support item
assignment
>>>
```

Tuples in Python

- Tuples are immutable objects, e.g. we CANNOT modify their contents.

Code

```
>>> my_t = (1, 10, 4, 5) # tuple creation ( ,)
>>> my_t[1]
10
>>> my_t = my_t[:2] + (9,) + my_t[3:] # Modifying the
third element
>>> my_t
(1, 10, 9, 5)
```

- Note that numbers and strings are immutable too.

Lists in Python

- Lists are mutable objects, e.g. we can modify their contents.

Code

```
>>> my_list = [1, 10, 4, 5] # List creation [ ]
>>> my_list[2]      # Accessing the third element
4
>>> my_list
[1, 10, 4, 5]
>>> my_list[2] = 9 # Modifying the third element
>>> my_list         # The modified list
[1, 10, 9, 5]
>>>
```

Be Careful

- Example:

Code

```
>>> lst1 = lst2 = [2,4,6,8]
>>> t1 = t2 = (1,3,7,9)
>>> lst1
[2, 4, 6, 8]
>>> lst2
[2, 4, 6, 8]
>>> t1
(1, 3, 7, 9)
>>> t2
(1, 3, 7, 9)
>>>
```

Be Careful

- Example:

Code

```
>>> t2 = t2[:2] + (5, ) + t2[3:] # Modifying the third
element t2
>>> lst2[2] = 5 # Modifying the third element of lst2
>>> t1
(1, 3, 7, 9)
>>> t2
(1, 3, 5, 9)
>>>
```

A change in t2 **does not** affect t1

Be Careful

- Example:

Code

```
>>> t2 = t2[:2] + (5, ) + t2[3:] # Modifying the third element t2
>>> lst2[2] = 5 # Modifying the third element of lst2
>>> t1
(1, 3, 7, 9)
>>> t2
(1, 3, 5, 9)
>>> lst1
[2, 4, 5, 8]
>>> lst2
[2, 4, 5, 8]
>>>
```

A change in t2 **does not** affect t1

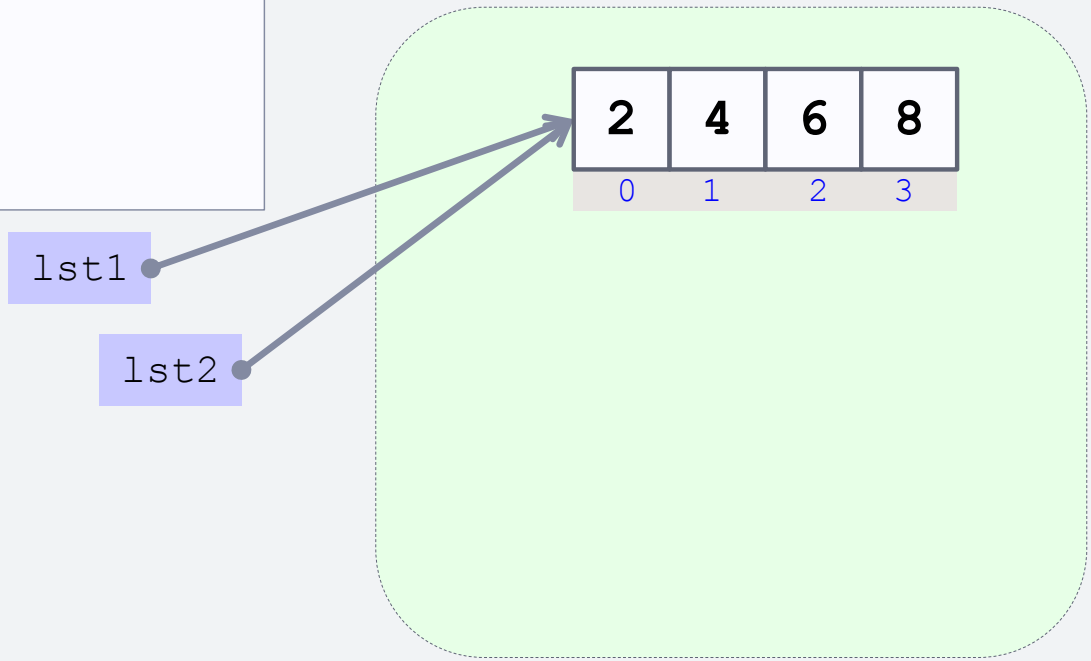
A change in lst2 **does** affect lst1

Be Careful

- State Diagram:

Code

```
>>> lst1 = lst2 = [2,4,6,8]
>>>
```

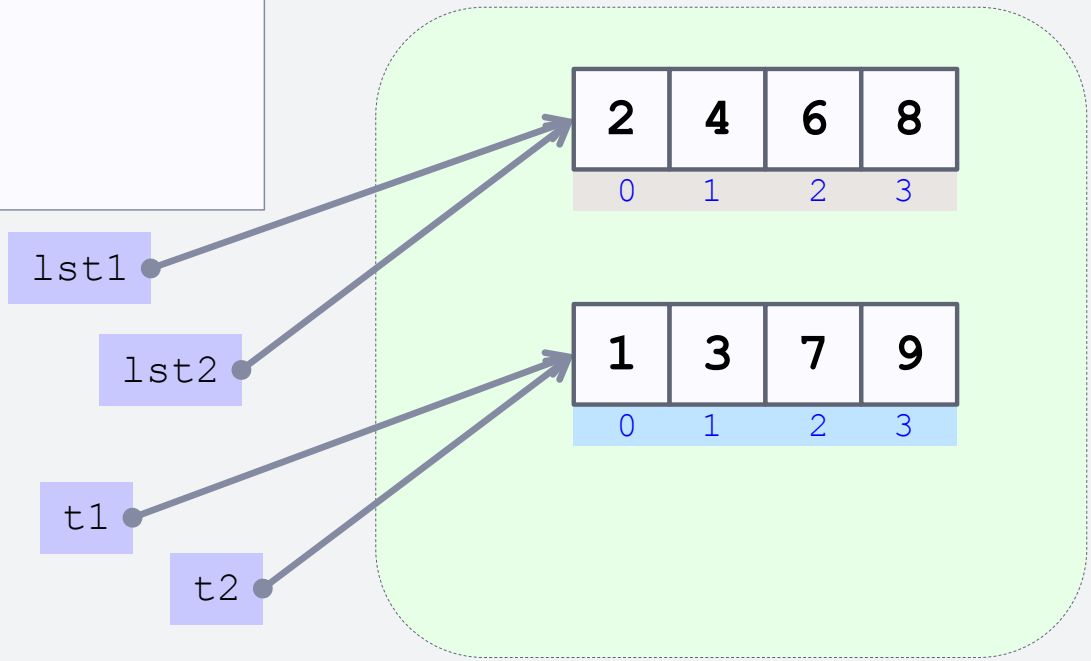


Be Careful

- State Diagram:

Code

```
>>> lst1 = lst2 = [2,4,6,8]
>>> t1 = t2 = (1,3,7,9)
>>>
```

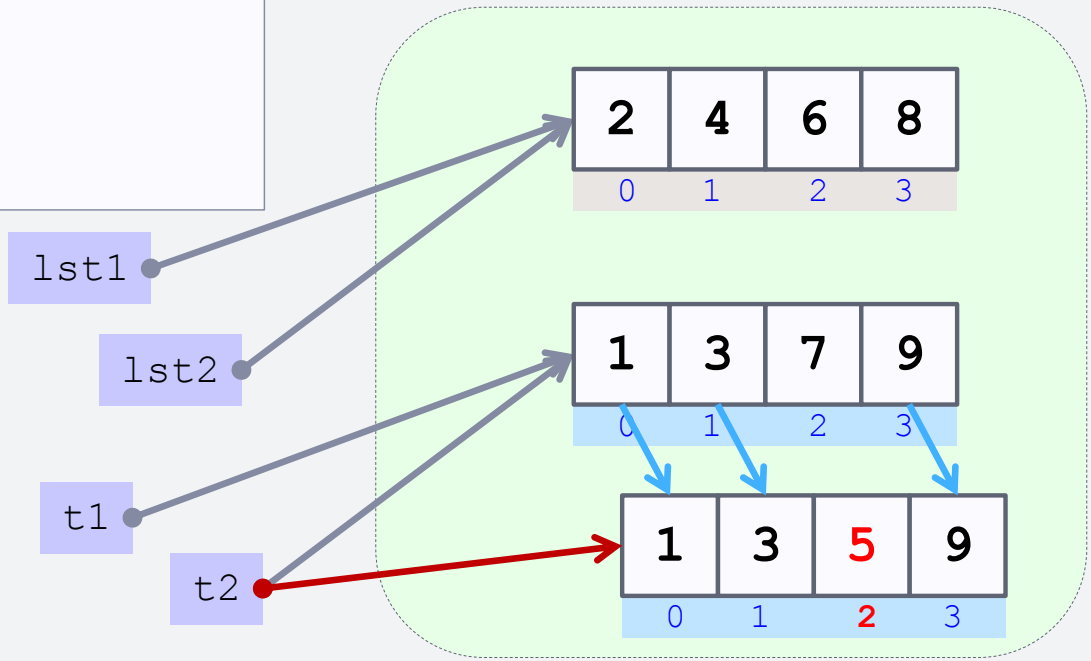


Be Careful

- State Diagram:

Code

```
>>> lst1 = lst2 = [2,4,6,8]
>>> t1 = t2 = (1,3,7,9)
>>> t2 = t2[:2] + (5, ) + t2[3:]
>>>
```

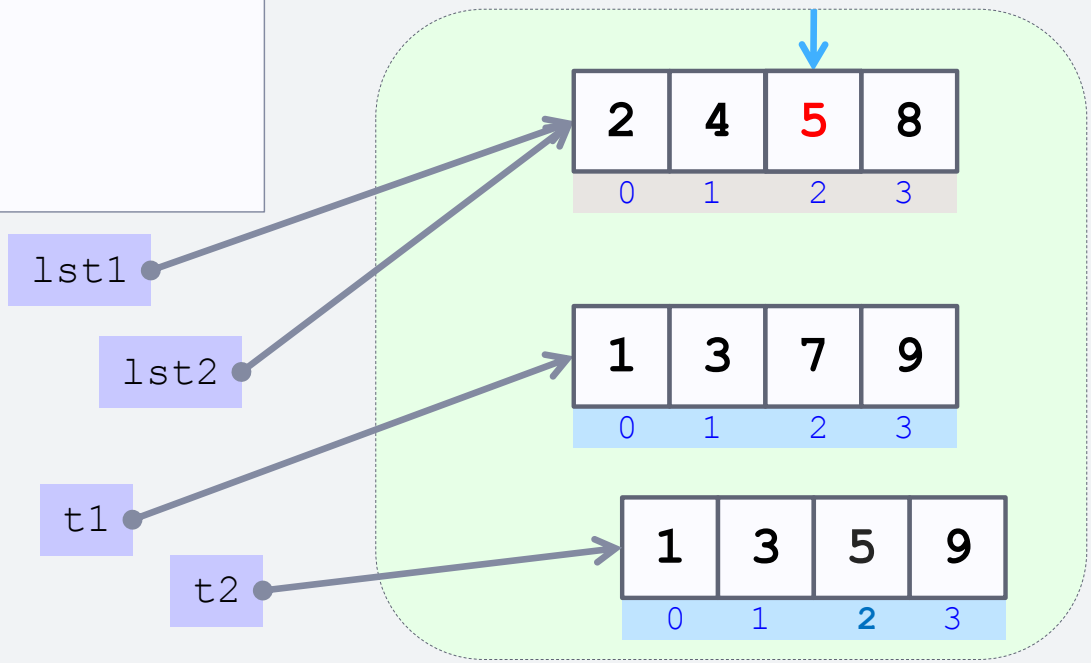


Be Careful

- State Diagram:

Code

```
>>> lst1 = lst2 = [2,4,6,8]
>>> t1 = t2 = (1,3,7,9)
>>> t2 = t2[:2] + (5, ) + t2[3:]
>>> lst2[2] = 5
>>>
```

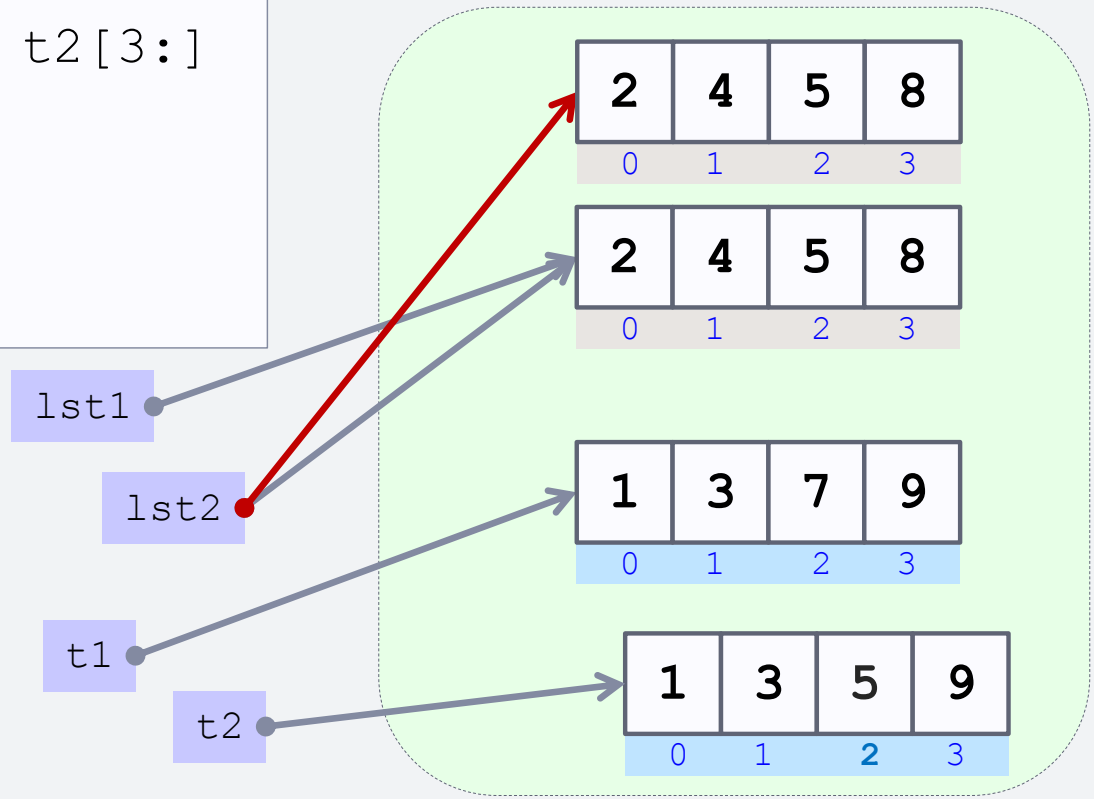


Be Careful

- State Diagram:

Code

```
>>> lst1 = lst2 = [2,4,6,8]
>>> t1 = t2 = (1,3,7,9)
>>> t2 = t2[:2] + (5, ) + t2[3:]
>>> lst2[2] = 5
>>>
>>> lst2 = [2, 4, 5, 8]
>>>
```

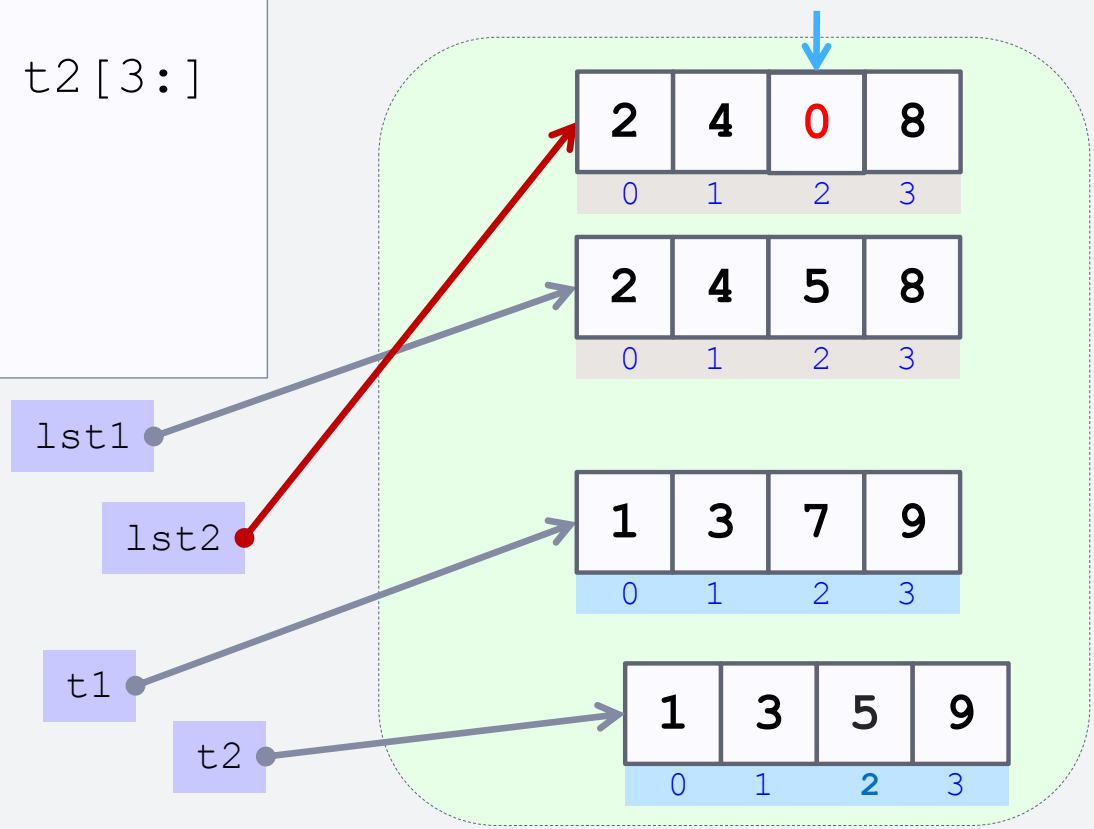


Be Careful

- State Diagram:

Code

```
>>> lst1 = lst2 = [2,4,6,8]
>>> t1 = t2 = (1,3,7,9)
>>> t2 = t2[:2] + (5, ) + t2[3:]
>>> lst2[2] = 5
>>>
>>> lst2 = [2, 4, 5, 8]
>>> lst2[2] = 0
```

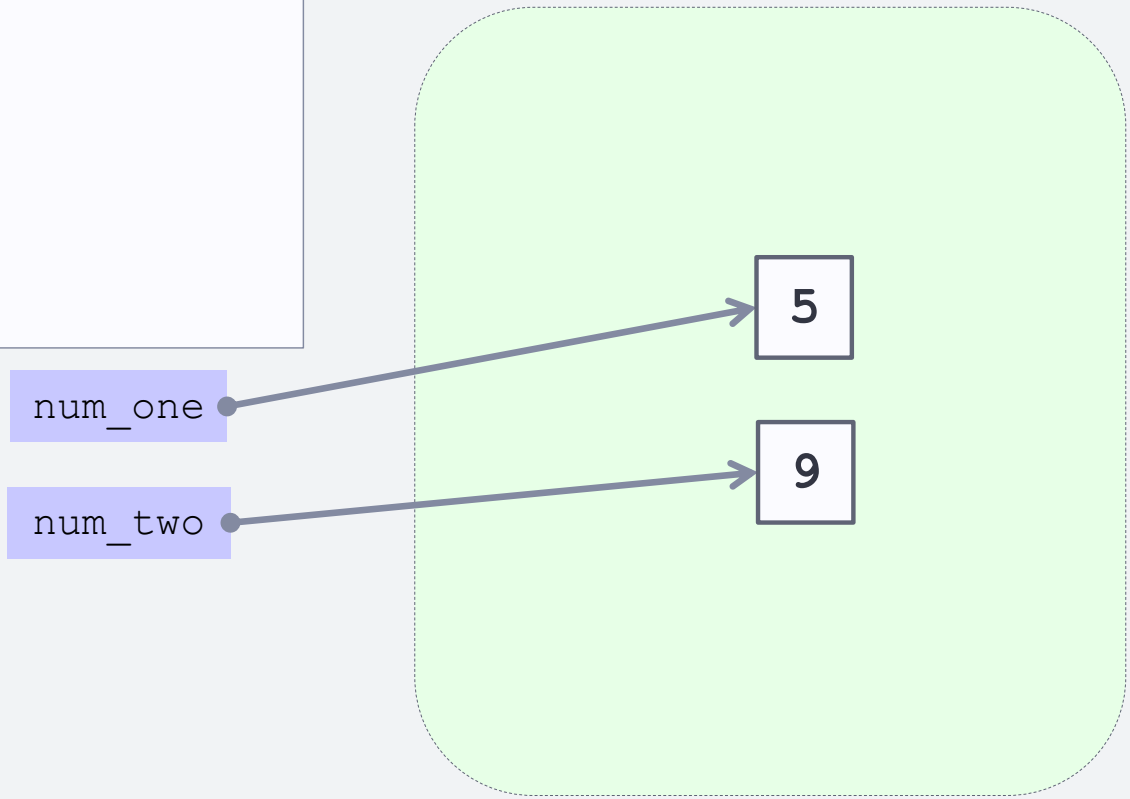


Swapping Values

- State Diagram:

Code

```
>>> num_one = 5
>>> num_two = 9
>>>
```

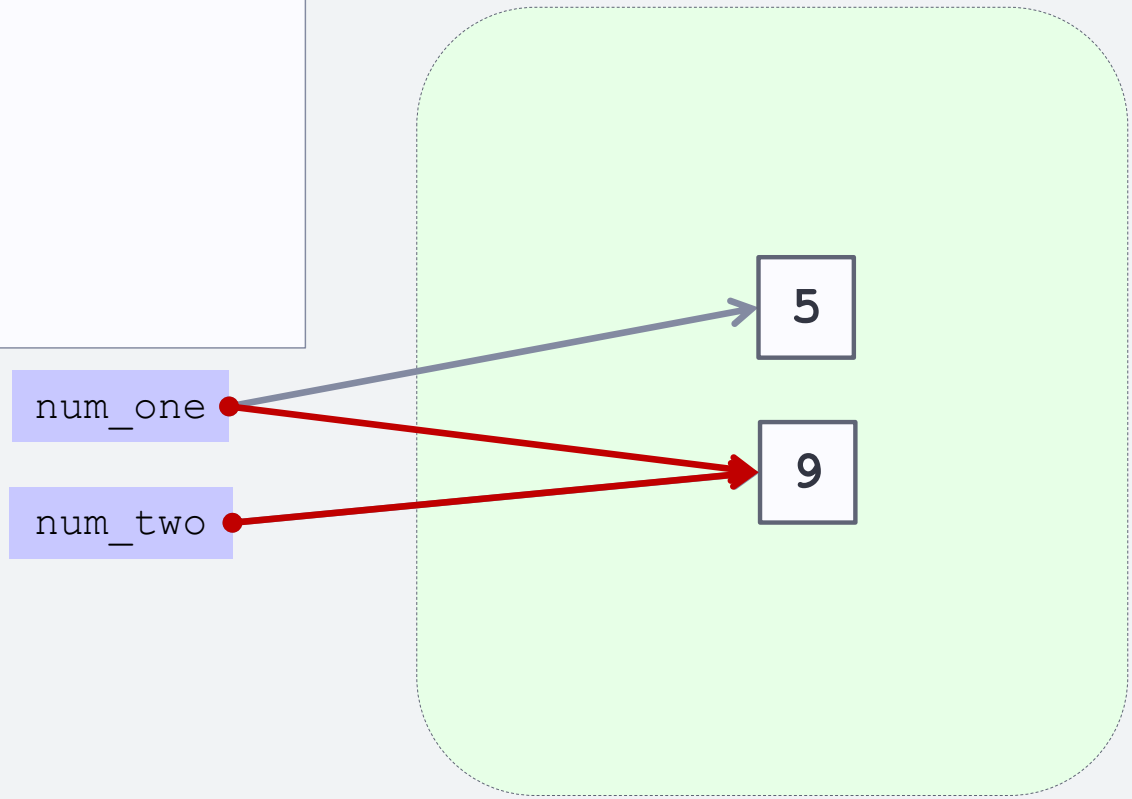


Swapping Values

- State Diagram: First Attempt

Code

```
>>> num_one = 5
>>> num_two = 9
>>> num_one = num_two
>>> num_two = num_one
>>>
```

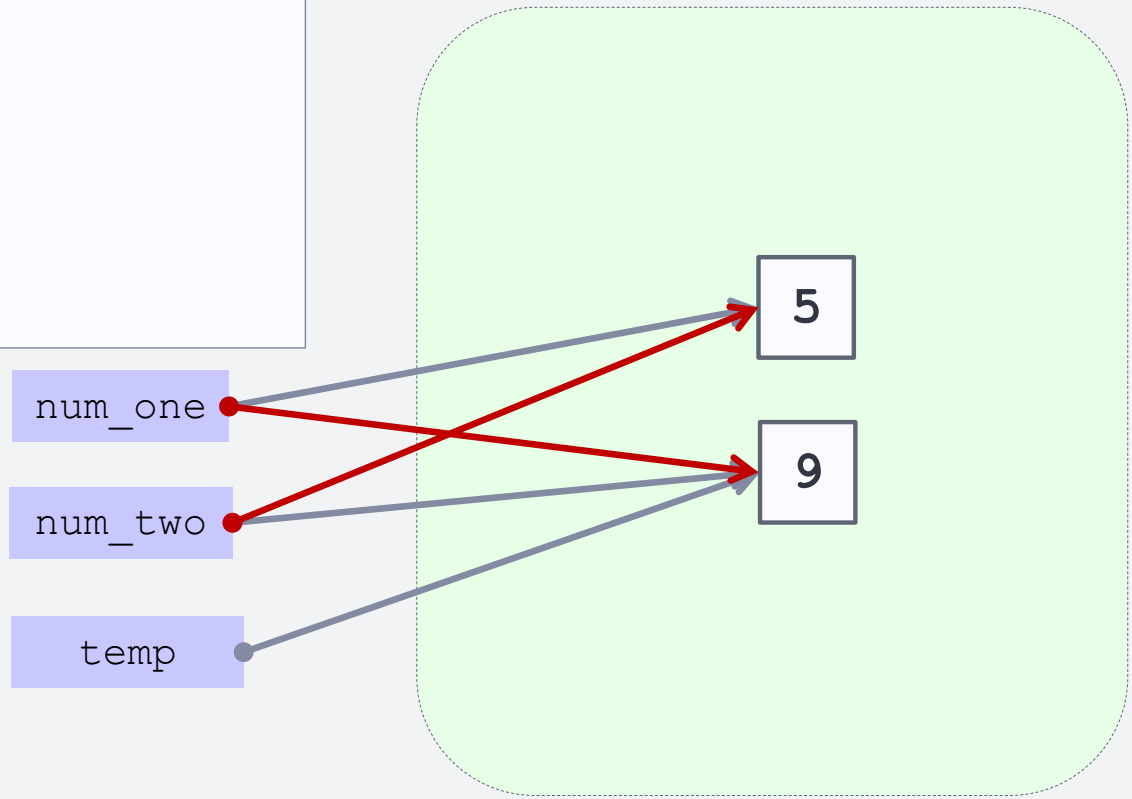


Swapping Values

- State Diagram: Second Attempt

Code

```
>>> num_one = 5
>>> num_two = 9
>>> temp = num_two
>>> num_two = num_one
>>> num_one = temp
>>>
```



Personal work

Investigate what is meant by SLICING strings, lists and tuples.

Iteration

DEFINITE
&
INDEFINITE
LOOPS

Iteration

- We need a structure to execute a sequence of statements multiple times in succession

- How can we proceed if we are not sure how many times it needs to be repeated?

Definite Loop: For statement

- The simplest kind of loop
- It will execute a definite amount of times
- The for loop statement

```
for <var> in <iterable>:
    <body>
```

  - Iterable can return its element one at a time
  - The body of the loop can be any sequence of Python statements
  - The variable after the keyword `for` is called the ***loop index***.

Example

Code

```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside loop: square of", str(val), "is",
          str(square_val)

print "After the for loop"
```

Python shell

```
Before the for loop
--> inside loop: square of 1 is 1
--> inside loop: square of 2 is 4
--> inside loop: square of 3 is 9
After the for loop
```

Example

Code

```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside . . ."

print "After the for loop"
```

Python shell

Example

Code

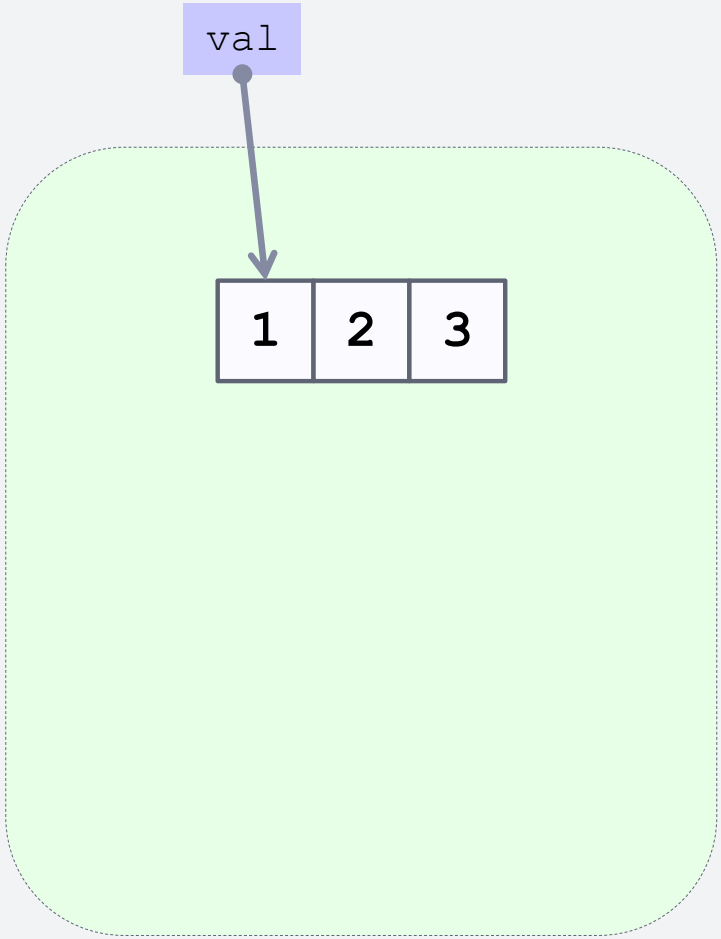
```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside . . ."

print "After the for loop"
```

Python shell

```
Before the for loop
```



Example

Code

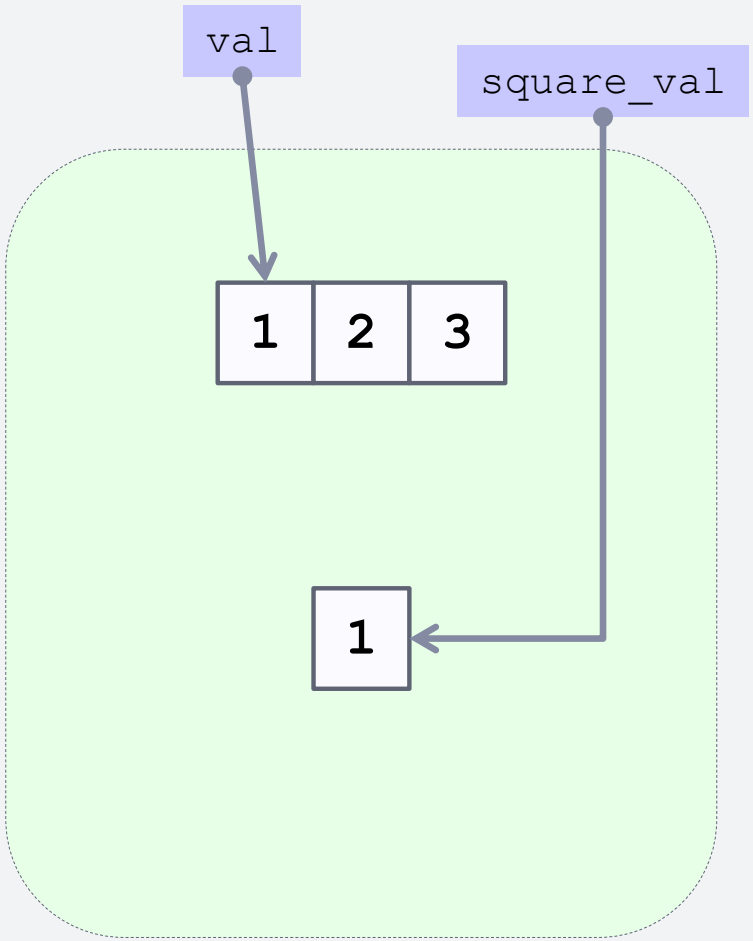
```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside . . ."

print "After the for loop"
```

Python shell

```
Before the for loop
```



Example

Code

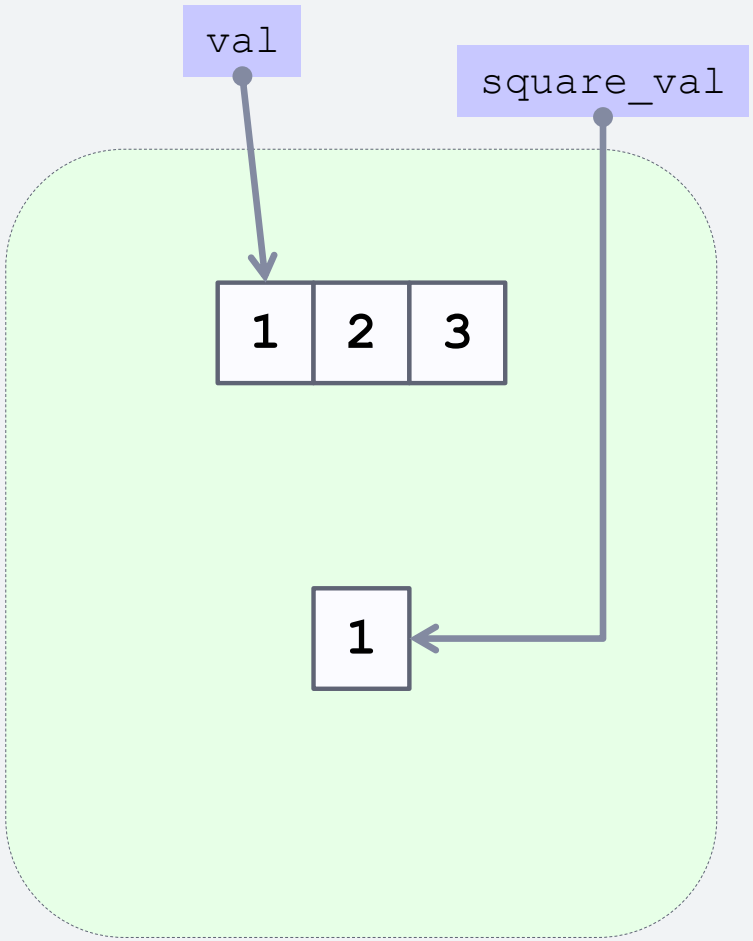
```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside . . ."

print "After the for loop"
```

Python shell

```
Before the for loop
--> inside loop: square of 1 is 1
```



Example

Code

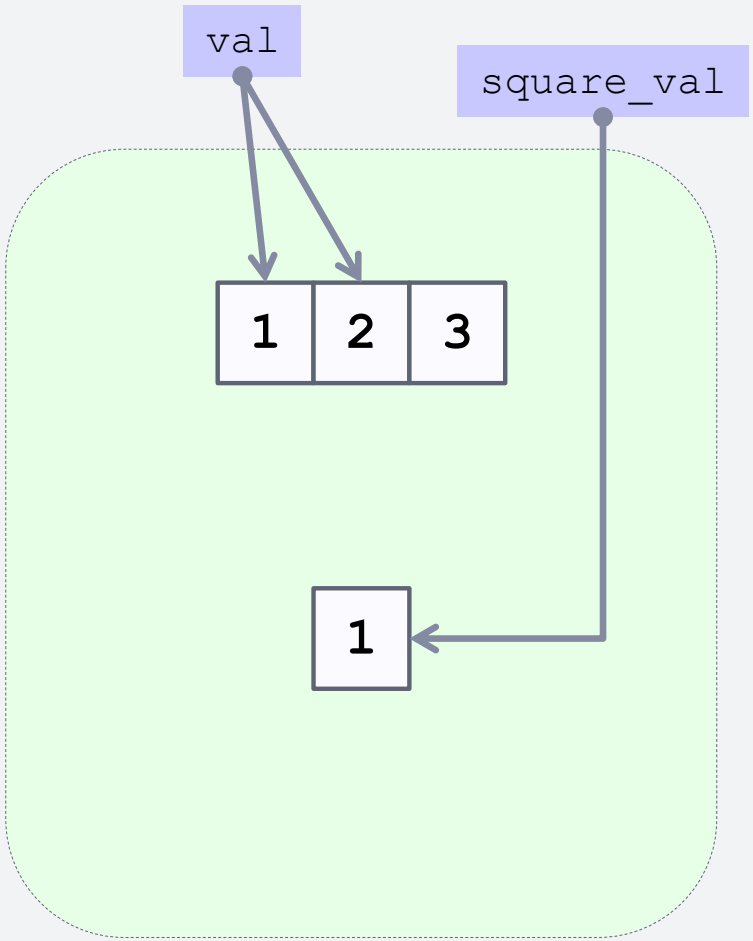
```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside . . ."

print "After the for loop"
```

Python shell

```
Before the for loop
--> inside loop: square of 1 is 1
```



Example

Code

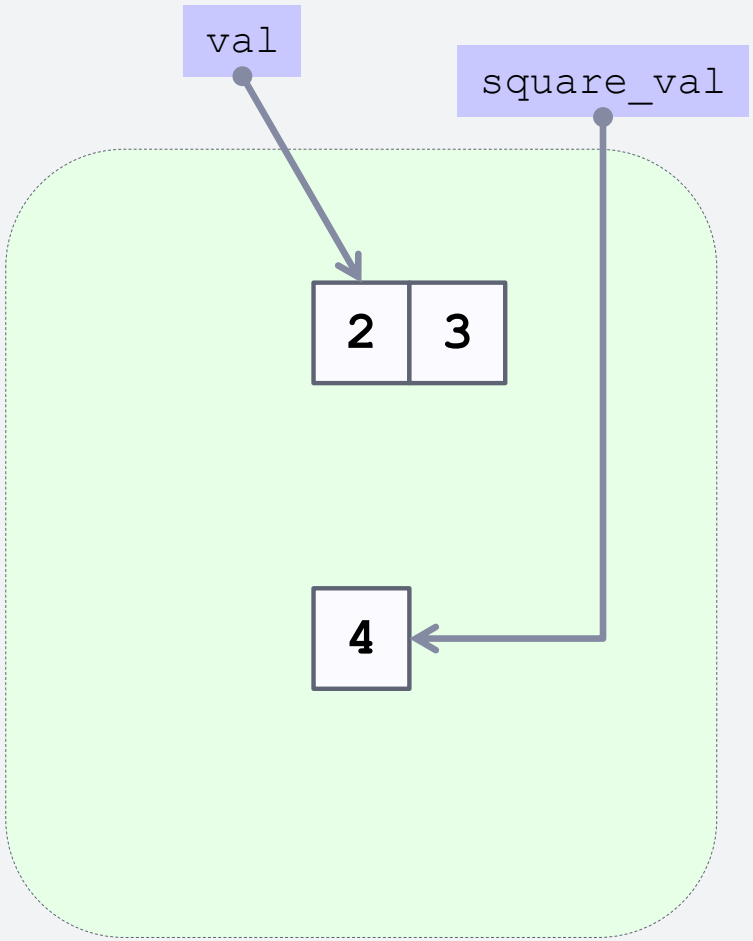
```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside . . ."

print "After the for loop"
```

Python shell

```
Before the for loop
--> inside loop: square of 1 is 1
```



Example

Code

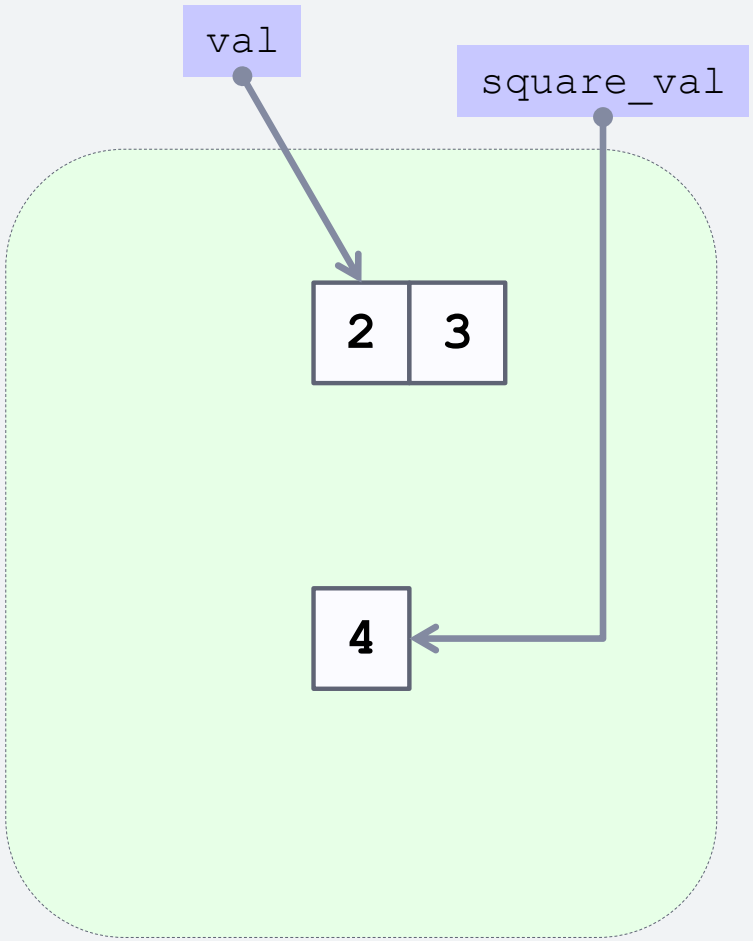
```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside . . ."

print "After the for loop"
```

Python shell

```
Before the for loop
--> inside loop: square of 1 is 1
--> inside loop: square of 2 is 4
```



Example

Code

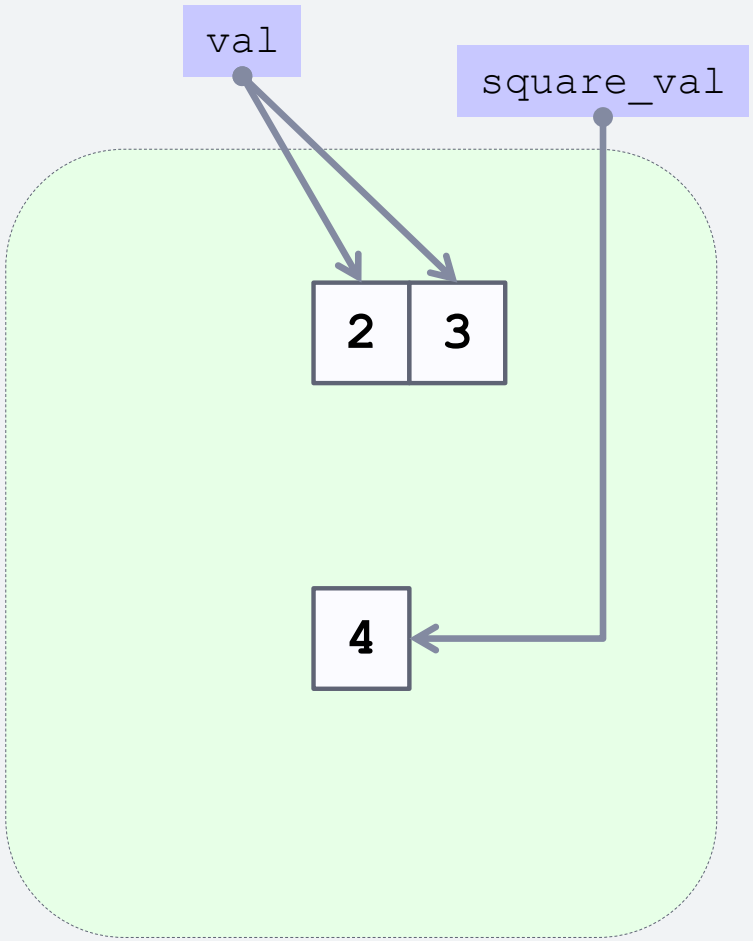
```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside . . ."

print "After the for loop"
```

Python shell

```
Before the for loop
--> inside loop: square of 1 is 1
--> inside loop: square of 2 is 4
```



Example

Code

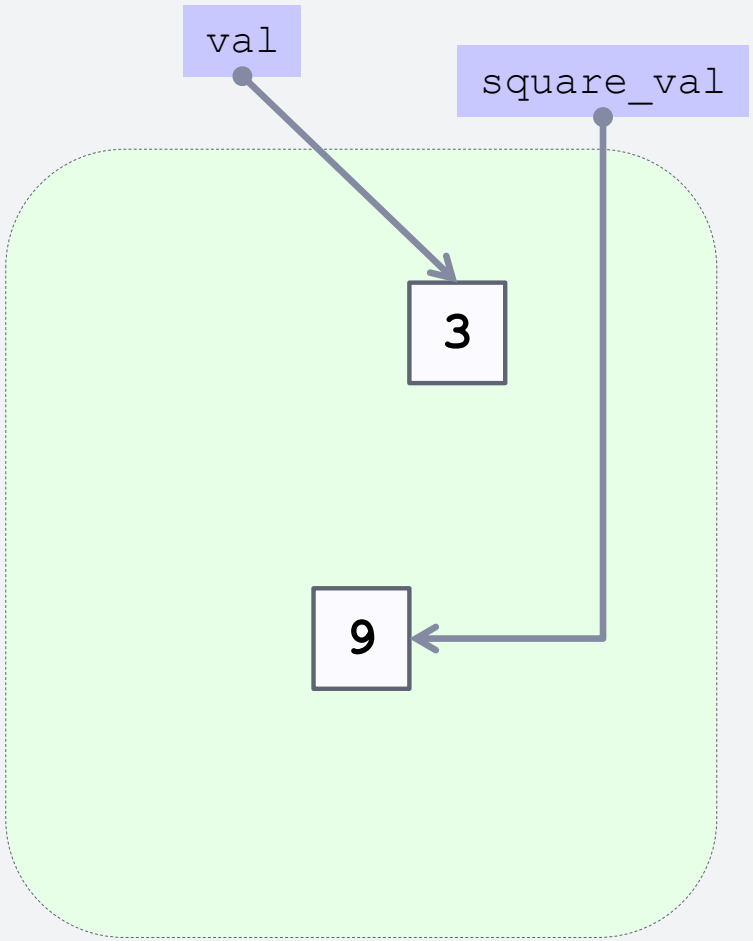
```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside . . ."

print "After the for loop"
```

Python shell

```
Before the for loop
--> inside loop: square of 1 is 1
--> inside loop: square of 2 is 4
```



Example

Code

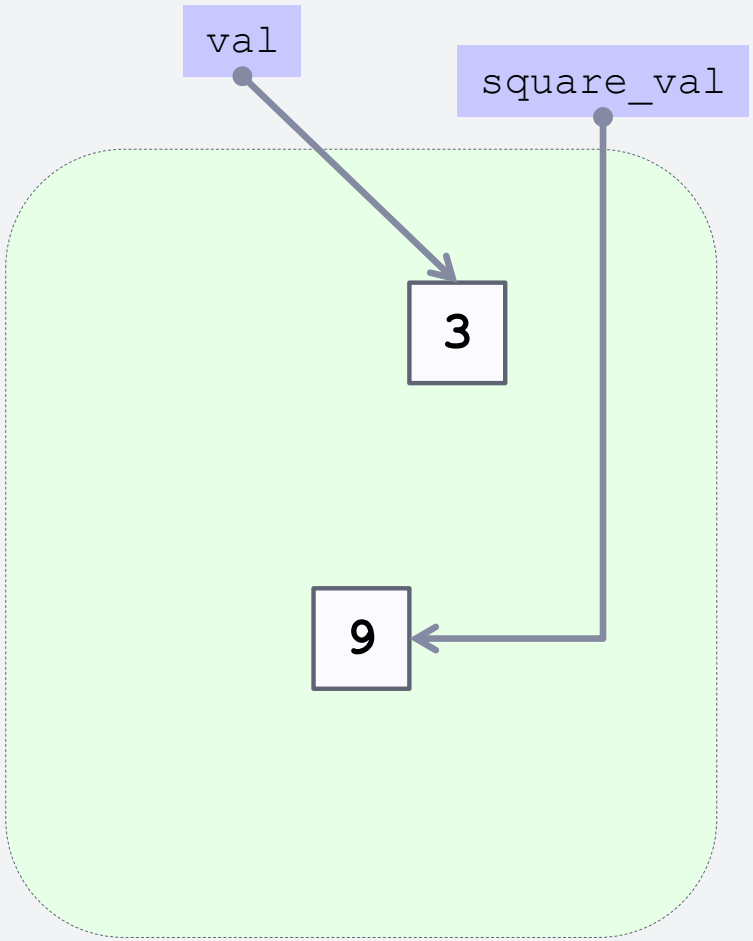
```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside . . ."

print "After the for loop"
```

Python shell

```
Before the for loop
--> inside loop: square of 1 is 1
--> inside loop: square of 2 is 4
--> inside loop: square of 3 is 9
```



Example

Code

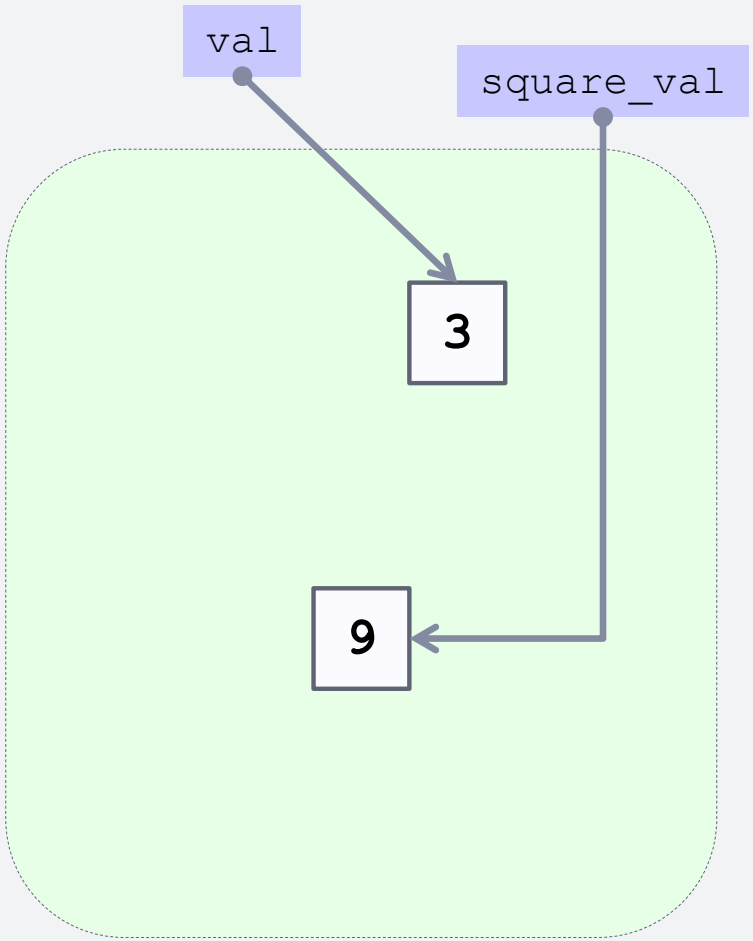
```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside . . ."

print "After the for loop"
```

Python shell

```
Before the for loop
--> inside loop: square of 1 is 1
--> inside loop: square of 2 is 4
--> inside loop: square of 3 is 9
After the for loop
```



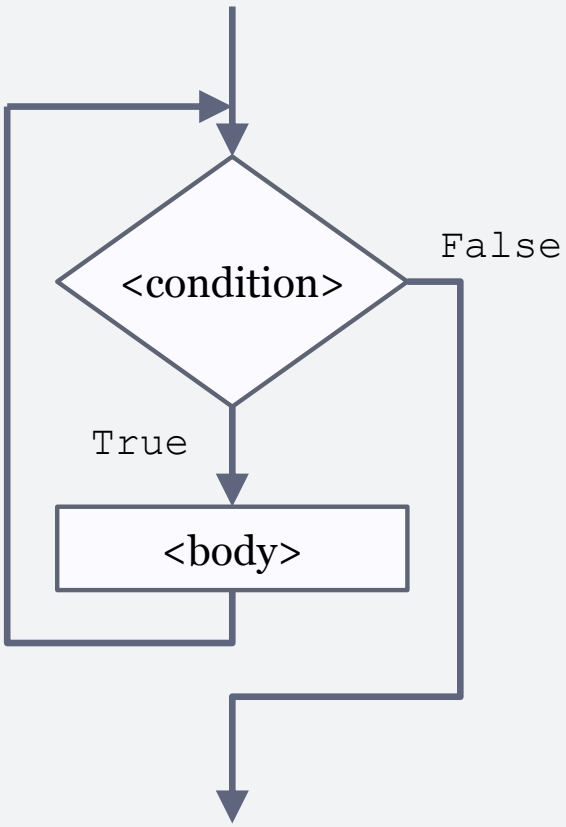
Indefinite Loop: While Statement

- An indefinite loop keeps iterating until certain condition are met (conditional loop)
- There is no guarantee ahead of time regarding how many times the loop will go around
  - zero time
  - x-times
  - indefinitely

Indefinite Loop: While Statement

- The while loop statement

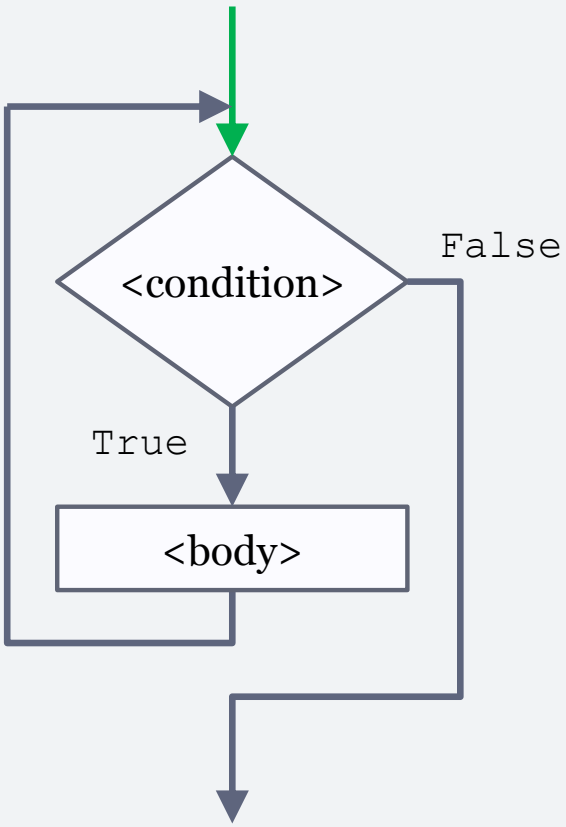
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

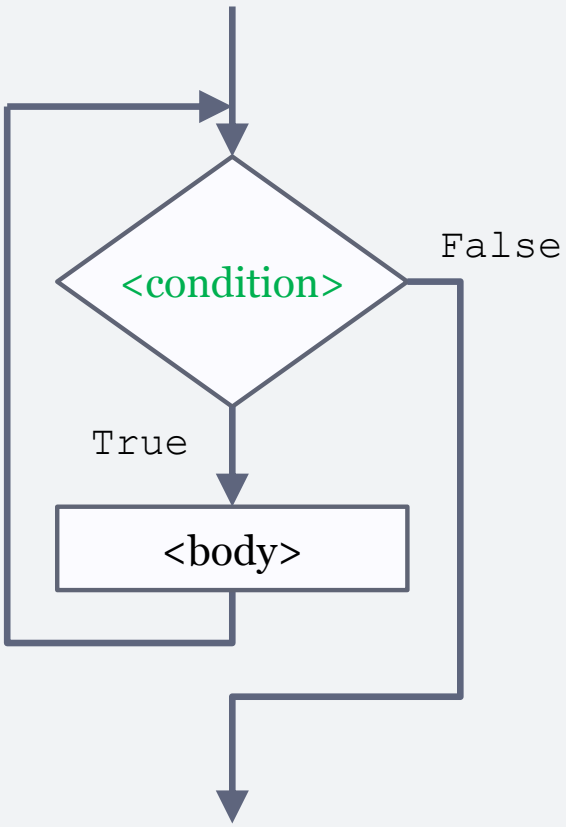
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

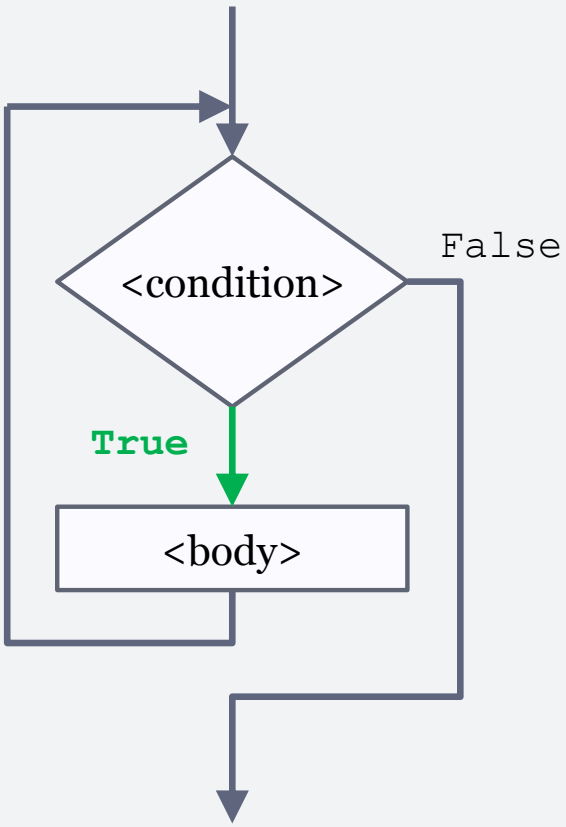
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

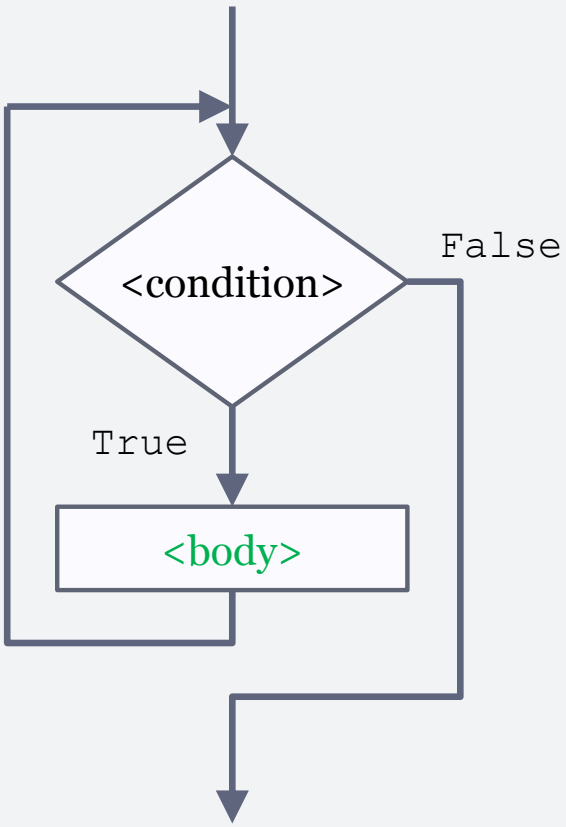
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

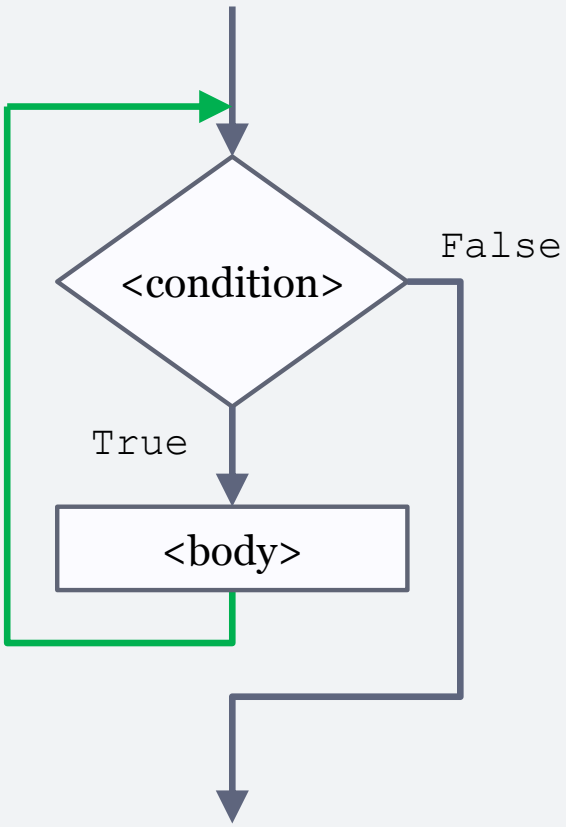
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

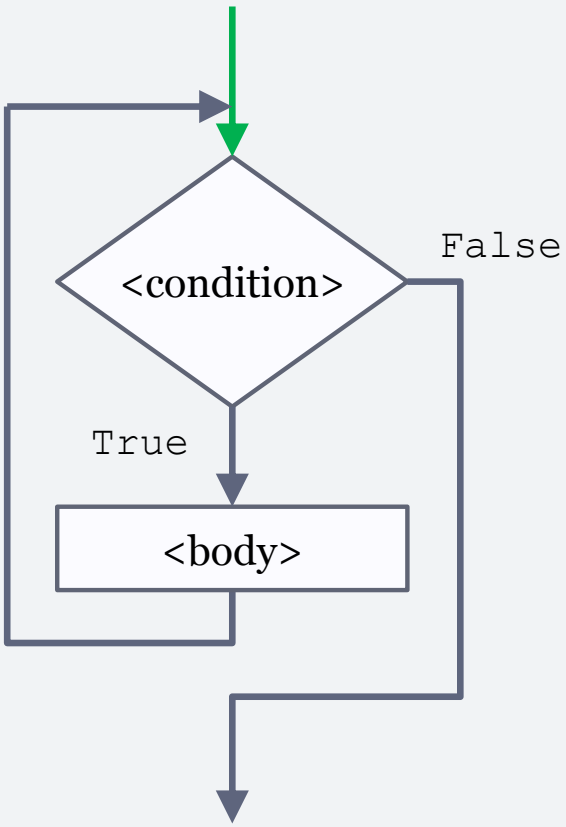
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

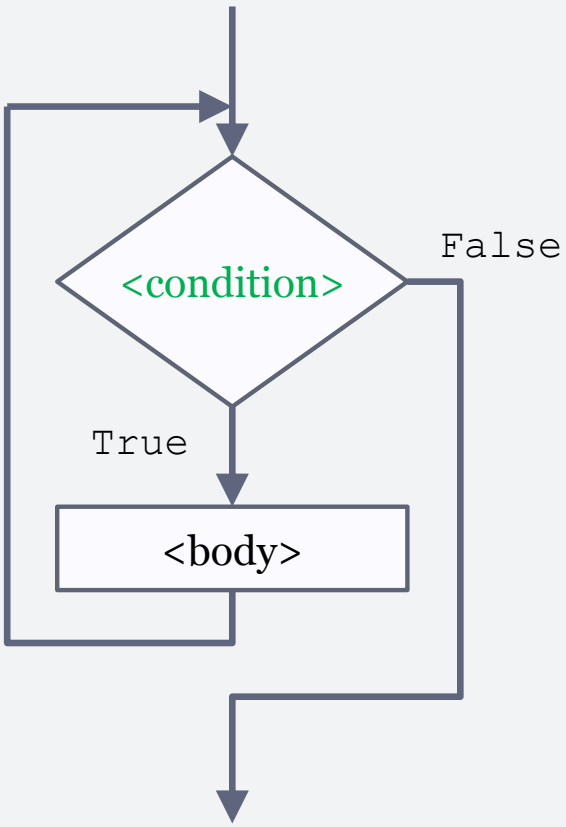
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

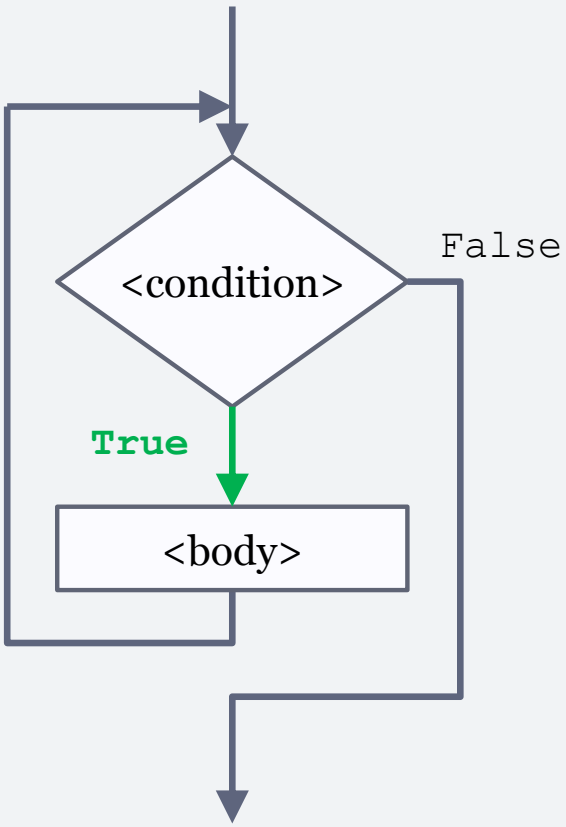
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

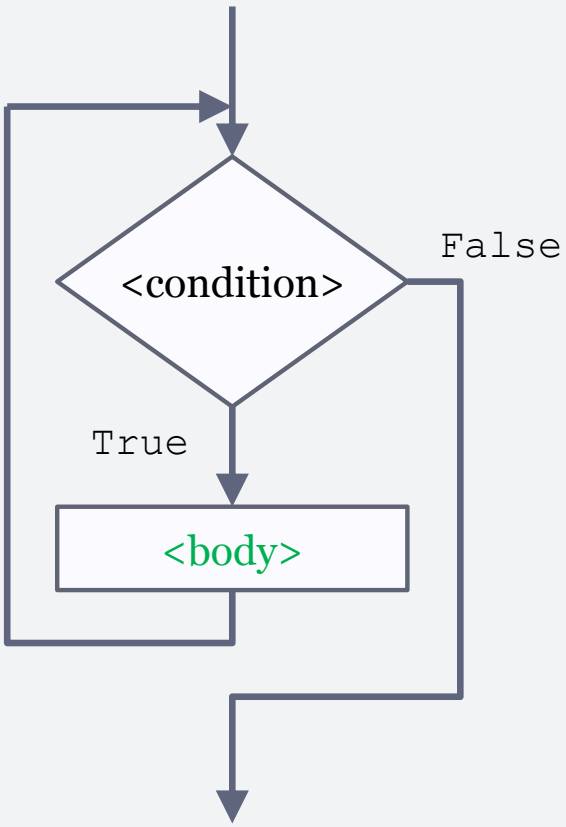
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

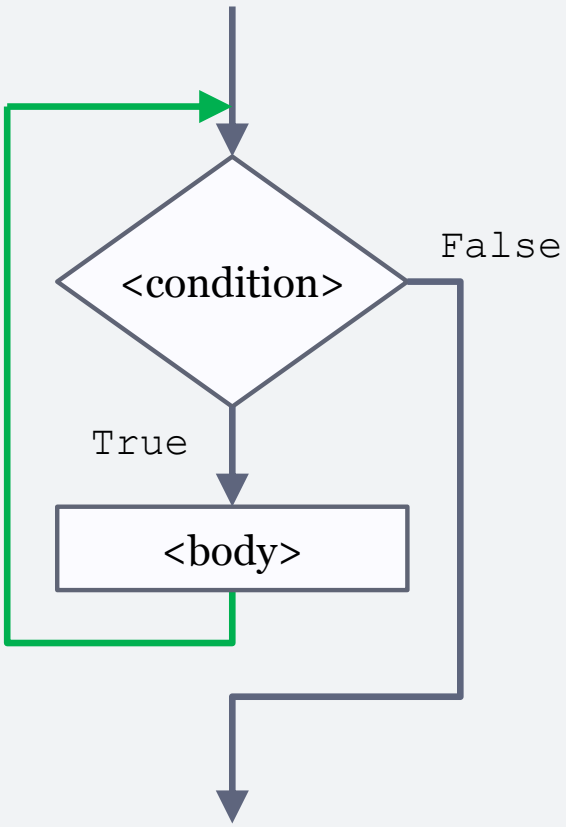
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

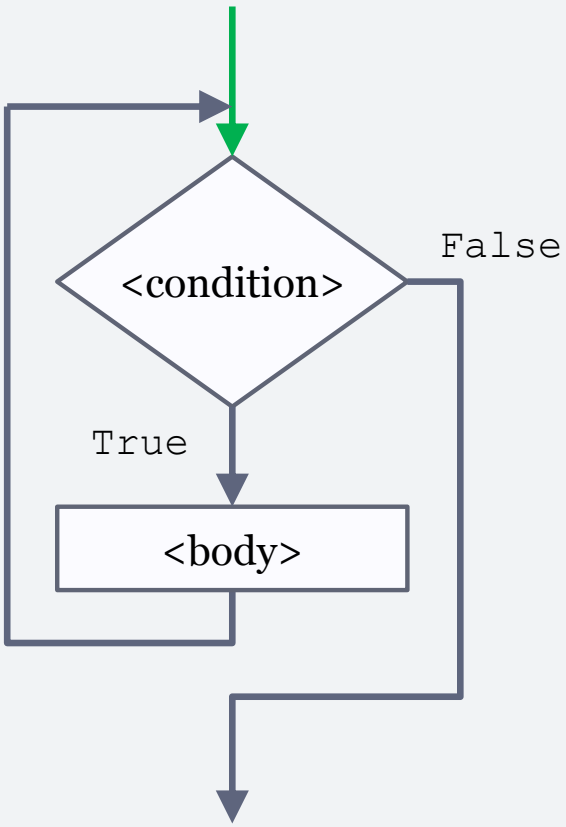
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

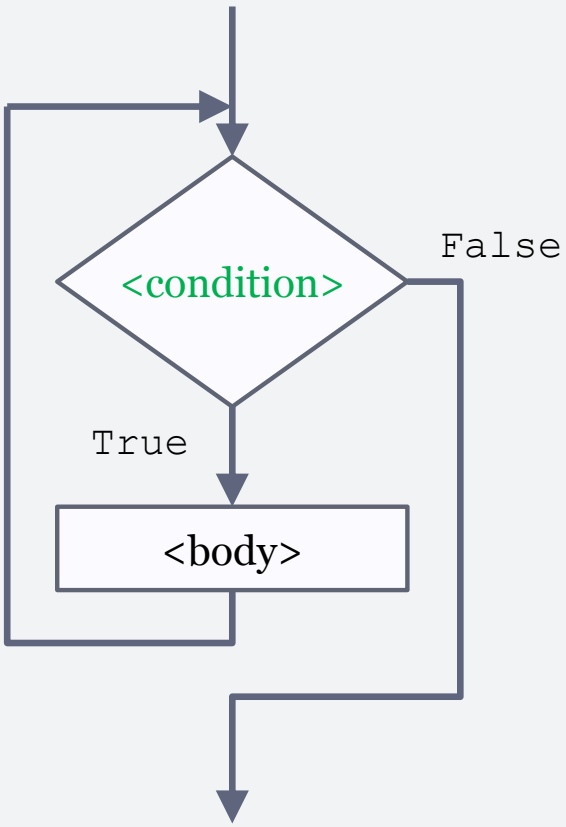
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

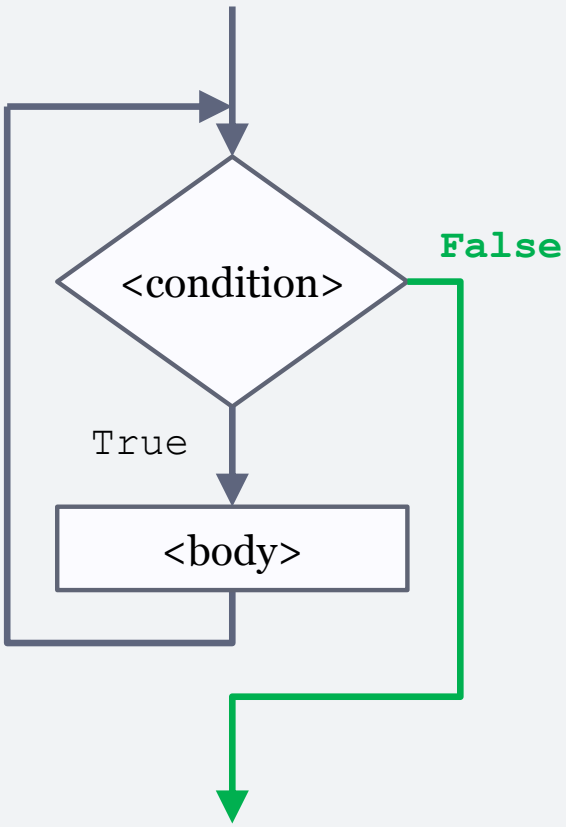
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

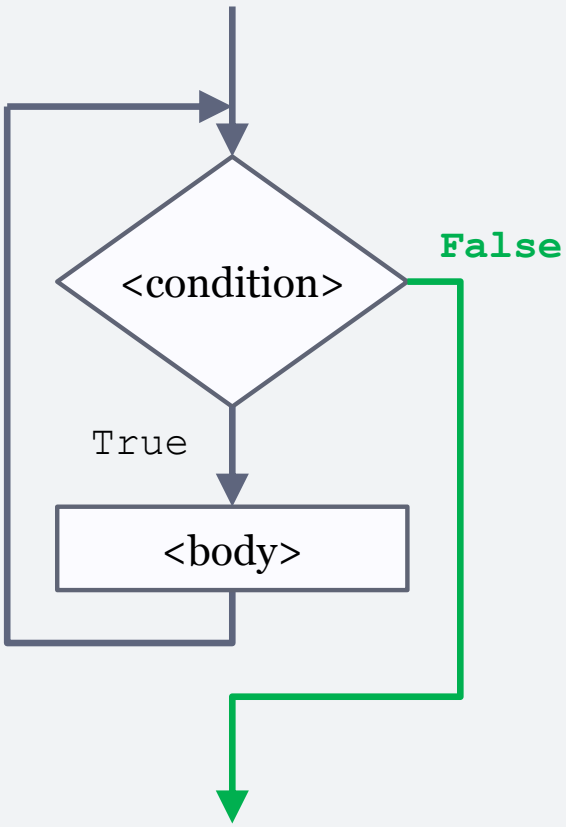
```
while <condition>:
    <body>
```



Indefinite Loop: While Statement

- The while loop statement

```
while <condition>:
    <body>
```

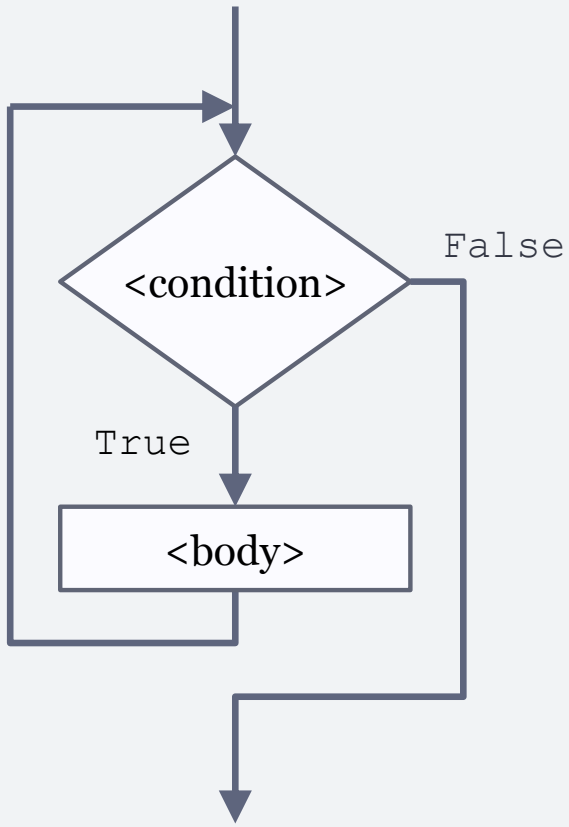


Indefinite Loop: While Statement

- The while loop statement

```
while <condition>:
    <body>
```

- The <body> usually contains statements that modify the evaluation of <condition> at some point



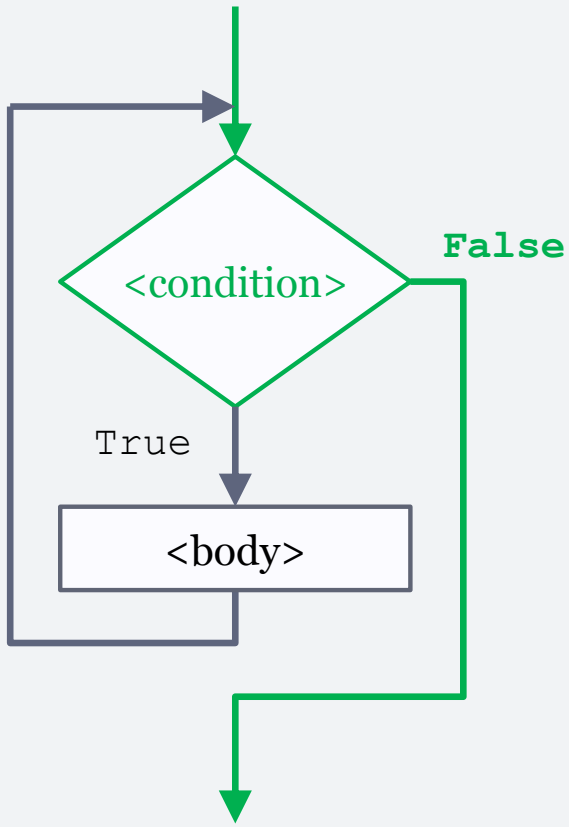
Indefinite Loop: While Statement

- The while loop statement

```
while <condition>:
    <body>
```

- The <body> usually contains statements that modify the evaluation of <condition> at some point

- The <body> may never be executed!

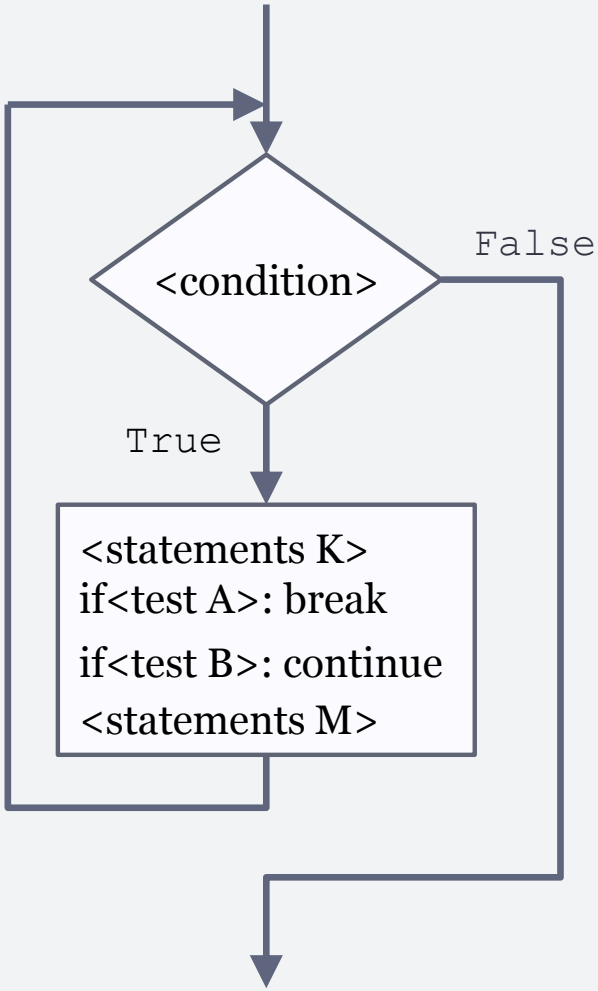


General Form

- The while loop general form
  - Two new Keywords:

1. **break**
2. **continue**

```
while <condition>:
    <statements K>
    if<test A>: break
    if<test B>: continue
    <statements M>
```

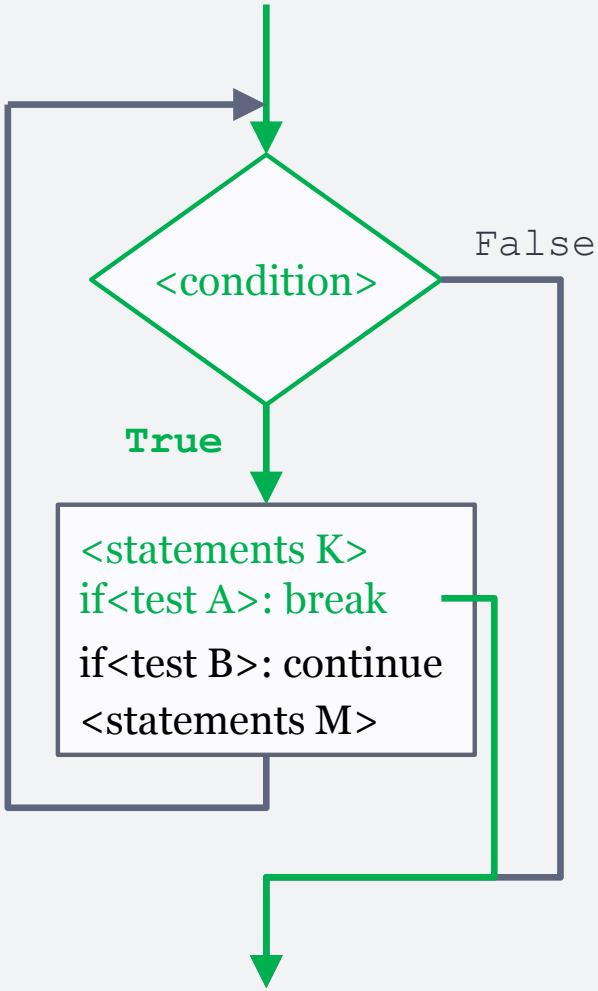


General Form

- The while loop general form
  - Two new Keywords:

    1. **break**
    2. **continue**

```
while <condition>:
    <statements K>
    if<test A>: break
    if<test B>: continue
    <statements M>
```



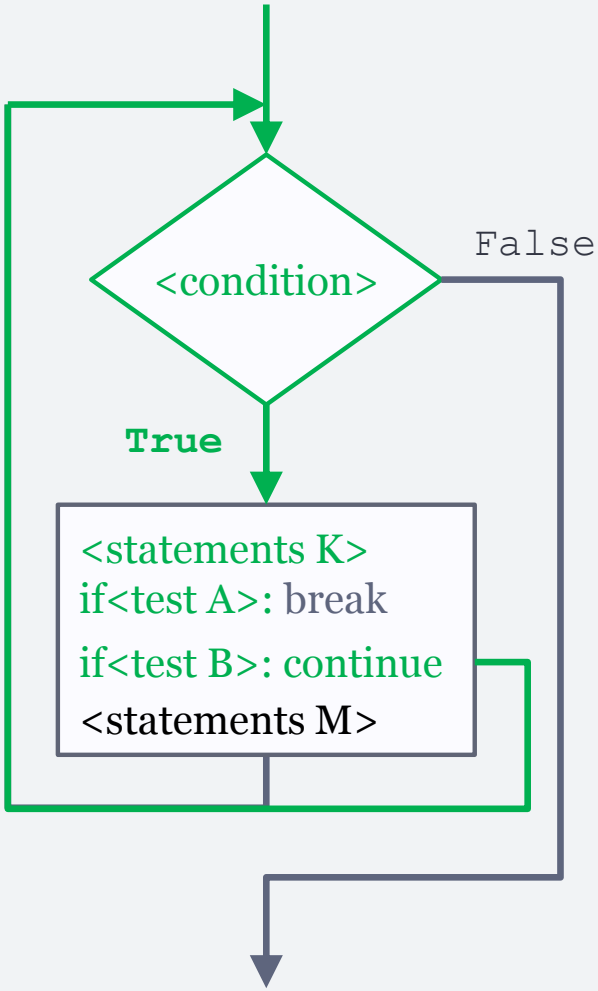
  - Test A is True

General Form

- The while loop general form
  - Two new Keywords:

    1. **break**
    2. **continue**

```
while <condition>:
    <statements K>
    if<test A>: break
    if<test B>: continue
    <statements M>
```



  - Test A is False & Test B is True

General Form

- The for loop general form

```
for <val> in <iterable>:
    <statements K>
    if<test A>: break
    if<test B>: continue
    <statements M>
```

Summary

- We have seen mutable and immutable data type
  - consequences when modifying data
- Definite iteration
  - for loops
- Indefinite iteration (a.k.a conditional loop)
  - while loops
- General form using `break` and `continue`

Next week

- Scope of variables and parameters
- Parameter passed by value or by reference
  - Understanding mutable and immutable object is essential

Exercises

- Consider a problem where we want to record and store the marks of many students, each students having more than one mark (several modules with same credits).
  - what form the data structure would look like
  - how would you write a program to enter the marks of one student
  - how would you proceed to enter the marks of more than one students
  - how would you calculate the average mark of each student
  - what if modules don't have the same credits (e.g. 10, 20, 30)

Exercises

- Calculate the average of values in a list
- Calculate the average of values entered by a user, we don't know how many values the user want to enter. The user should enter an empty value to signal the end of data input, then a result must be returned.