How many iterations?

The number of iterations of a for loop is

\[ \text{upper\_bound} - \text{lower\_bound} \]

```python
count = 0
# 59 - 2 = 57 iterations
for num in range(2, 59):
    count += 1

print("count is", count)
```
Loop Target Variable

- Don’t try to change the loop variable inside the loop!

```python
total = 0
for num in range(10) :
    total += num    # OK
    num += 1        # Bad !!!
```
Nested Loops: How many iterations?

# Entire inner loop executes each time the outer loop executes

```python
count = 0
for y in range(10):
    for x in range(20):
        count += 1

Print("count is", count)  # count is 200
```
Nested Loops: How many iterations?

# count is reset to 0 each time the
# outer loop executes

for y in range(10) :
    count = 0
    for x in range(20) :
        count += 1

Print("count is", count)  # count is 20
Loop Target Variables

# Use a different variable name for each target variable in
# a nest of loops

count = 0
different
for y in range(10) :
    for x in range(20) :
        count += 1

# OK to reuse target variable in new loop
for y in range(10) :
    print(y)
Example: Print a table of sums from 1 to 10

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How to Design Large Programs

• Figure out the different meaningful pieces and how they interact with each other
• Implement each piece as a function
  – High level design is done first
  – Details are postponed until later
  – Functions can be independently implemented, even by different people!
My Adventure Game with Functions

Main Menu

Places

Challenges

Final outcome

main()

cave()
maze()

fight(enemy)
solve_riddle(tries)

struggle_free(tries)

die(message, duration)

win(message)
Passing Arguments to Functions

A Parameter in the function definition indicates the function expects to be passed an argument.

- A parameter looks like a variable

```python
def main():
    show_double(5)

def show_double(num):
    # num is a parameter
    print(num * 2)
```
Passing Arguments to Functions

- **Argument**: piece of data that is sent into a function
  - It’s as if the argument is *copied* at call time

```python
def main():
    show_double(5)

def show_double(num):
    print(num * 2)
```

(num = 5)
Passing Arguments to Functions

• You can call the same function more than once with different arguments

```python
def main():
    show_double(3)  # function prints 6
    show_double(5)  # function prints 10

def show_double(num) :
    print(num * 2)
```
Using Parameters in Functions

A Parameter is created when the function is called, and disappears when the function exits.

```python
def main():
    show_double(5)

def show_double(num):
    num = num * 2
    print(num)
```
Using Parameters In Functions

Changes made to a parameter within the function do not affect the argument outside the function

```python
def main():
    my_num = 5
    show_double(my_num)
    print(my_num)  # prints 5

def show_double(num):
    num = num * 2
    print(num)  # prints 10
```
Local Variables

• **Local variable**: variable that is assigned a value inside a function
  – Disappears when the function exits

```python
def show_double(num) :
    tmp = num * 2
    print(tmp)
```
Local Variables

• Different functions may have local variables with the same name
  – Functions do not see other function’s local variables

```python
def show_double(num):
    tmp = num * 2
    print(tmp)

def show_square(num):
    tmp = num * num
    print(tmp)
```
Passing Multiple Arguments

• Python allows writing a function that accepts multiple arguments

• Arguments are passed by position to corresponding parameters
  – First parameter receives value of first argument, second parameter receives value of second argument, etc.
def main():
    show_sum(5, 7)

def show_sum(num1, num2):
    tmp = num1 + num2
    print(tmp)
Variable Scope

- **Scope**: the part of a program in which a variable may be accessed
  - For local variable: function in which created
  - For parameters: the function in which it is declared

- Scoping is necessary for modularization
  - Prevents confusion and errors
  - Controls the boundary between the function and the “outside”