Turtle Graphics

Conditional Statements, Loops

#3

Serdar ARITAN

Biomechanics Research Group,
Faculty of Sports Sciences, and
Department of Computer Graphics
Hacettepe University, Ankara, Turkey

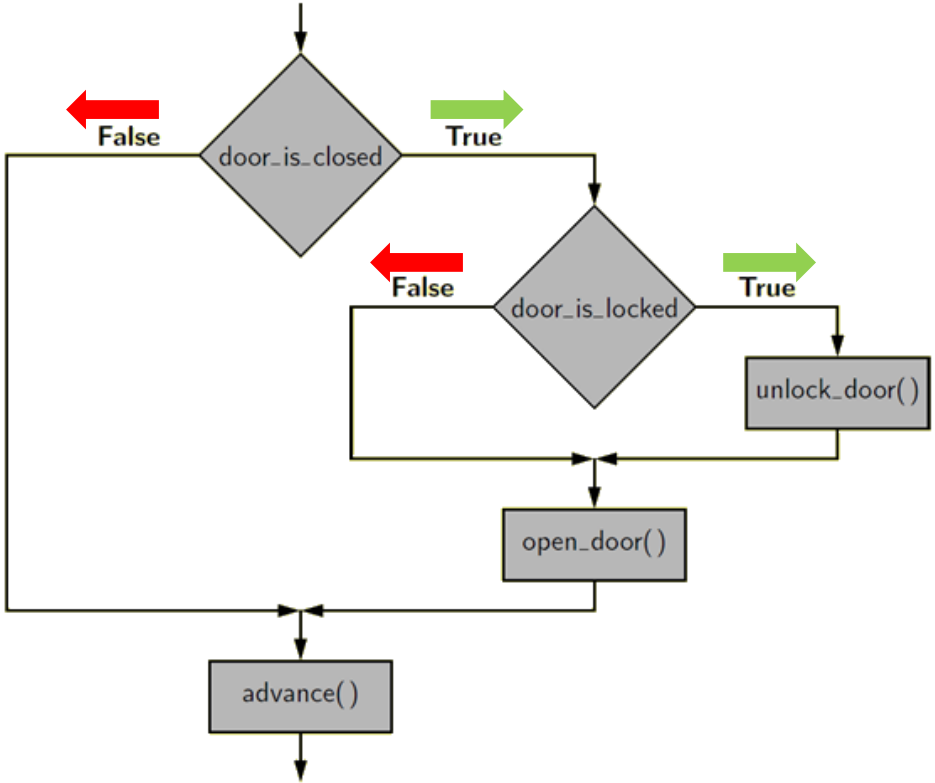
Conditional Statements

Conditional Statements

If This is True

```
if door is closed:
Do This-> if door is locked:
            Do This-> unlock door( )
                      open door( )
advance()
```

This is not true. It is False



Conditional Statements

As a programmer you will often find yourself needing to use a statement based on a condition... **if/else branches**

Example:

```
# age example 1
age = int(input("How old are you? "))
if age < 13:
     print ("Your age is ", age, "you are a children.")
else:
     print ("Your age is ", age, "you are an adult.")
```

Output:

```
>>>
How old are you? 5
your age is 5 you are a children
>>>
How old are you? 55
your age is 55 you are a adult
```

Conditional Statements

if/elif/else branches

Example:

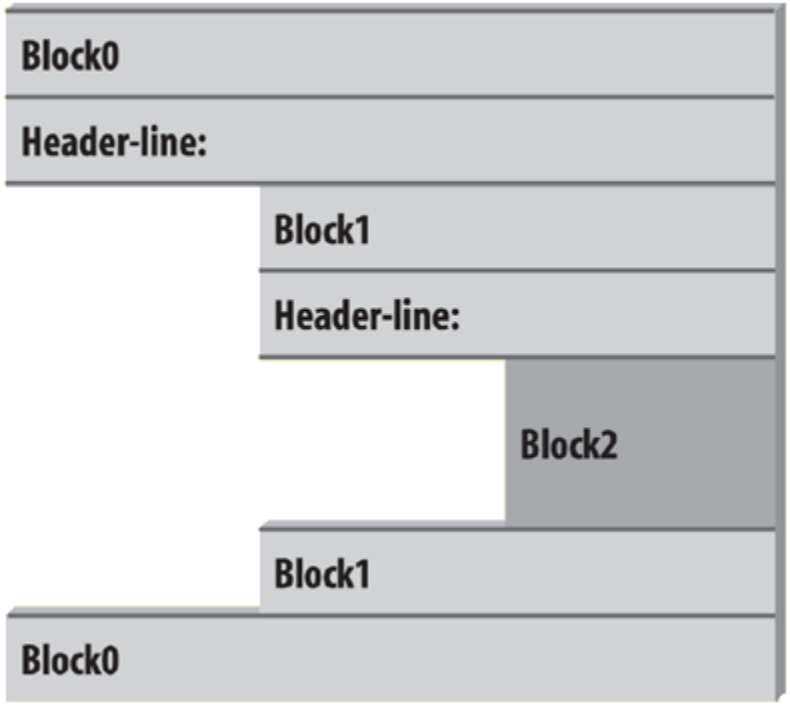
```
# age example 2


age = int(input("How old are you? "))

if age <= 13:
     print ("Your age is ", age, "you are a children.")
elif age >13 and age < 20
     print ("Your age is ", age, "you are a teenager.")
else:
     print ("Your age is ", age, "you are an adult.")
```

What is wrong with this code?

Conditional Statements

Nested blocks of code

Python detects block boundaries automatically, by line *indentation*—that is, the empty space to the left of your code.

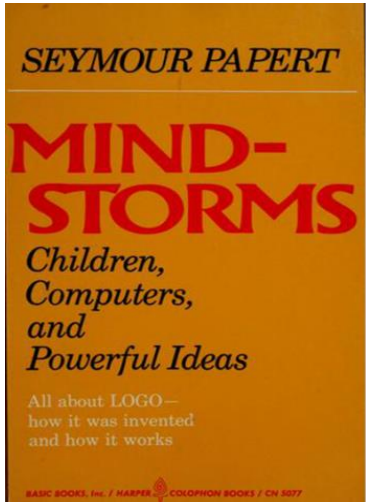



```
x = 1
if x:
    y = 2
    if y:
        print('block2')
    print('block1')
print('block0')
```

This code contains three blocks: the **first** is not indented at all, the **second** is indented four spaces, and the **third** is indented eight spaces.

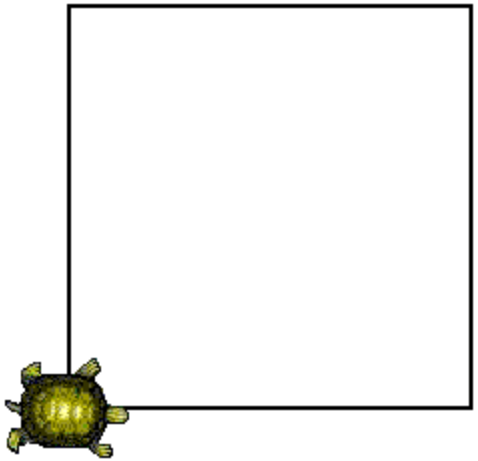
Turtle Graphics

Seymour Papert at MIT developed the Logo language and used it to teach programming to children, who used it to maneuver a robotic "turtle" that could make drawings on paper. In 1980, Professor Papert wrote a wonderful book called Mindstorms that describes his experiences with the Logo turtle and offers several important insights into the dynamics of learning.



Turtle Graphics

The individual commands consist of a single letter, which are usually followed by a number. For example, the command F120 asks the turtle to move forward 120 pixels in the direction it is facing. The command L90 asks the turtle to turn left 90 degrees. A program is simply a sequence of these commands.



The program
F120 L90 F120 L90 F120 L90 F120 L90

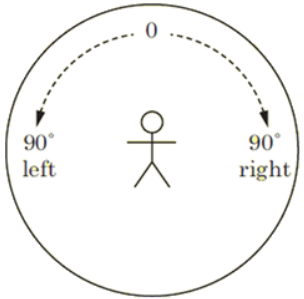
Pen Plotter

Pen Plotter

Laser Cutter

Turtle Movement Commands

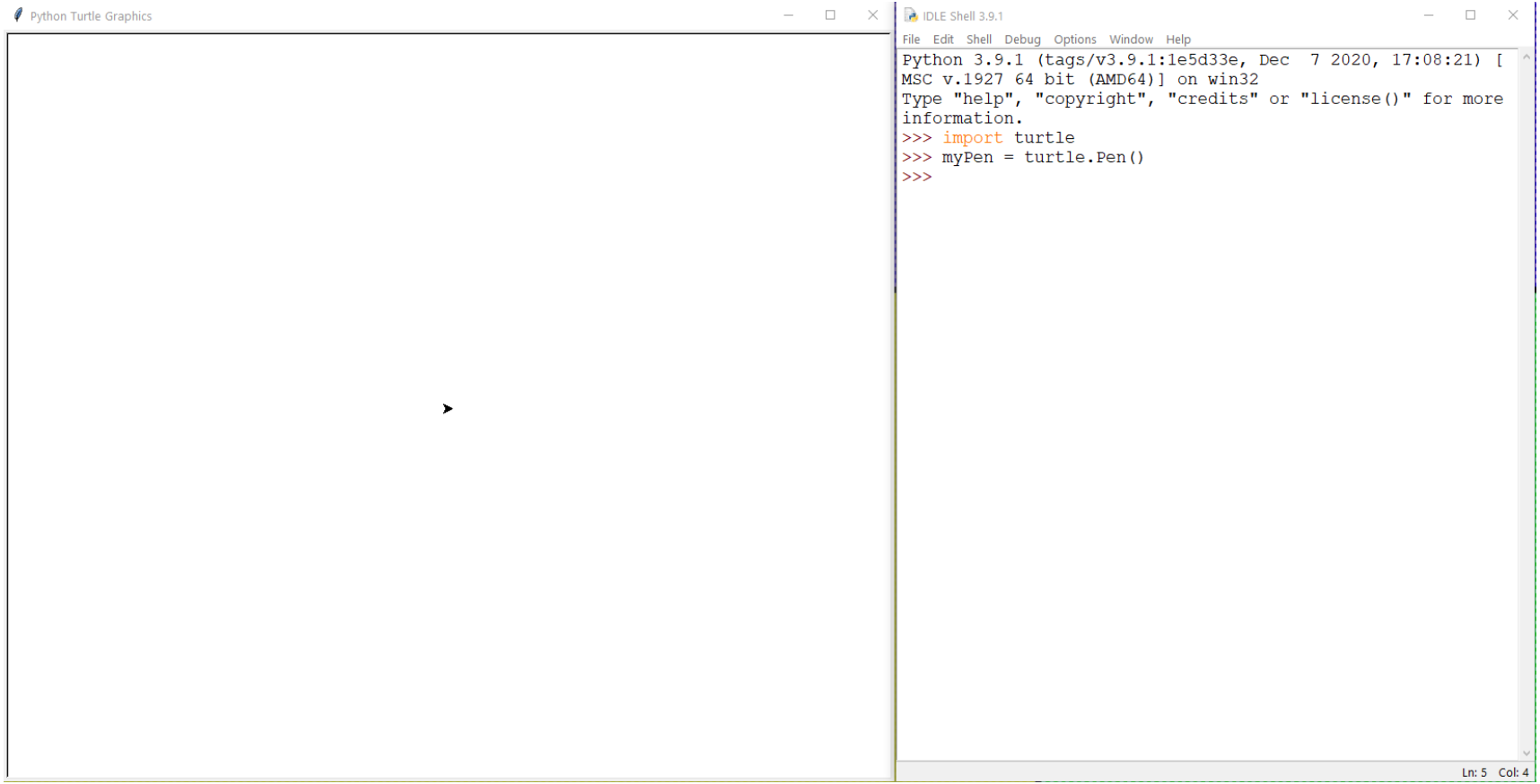
forward(*distance*)	Move forward *distance* in current direction.
backward(*distance*)	Move backward *distance* in the opposite direction.
right(*angle*)	Turn right by *angle* units.
left(*angle*)	Turn left by *angle* units.
goto(*x*,*y*)	Move turtle to absolute screen position (*x*,*y*).
home()	Move turtle to origin (0,0), facing the default direction (typically east).
speed(*speed*)	Set turtle drawing *speed* as int in range 0–10.



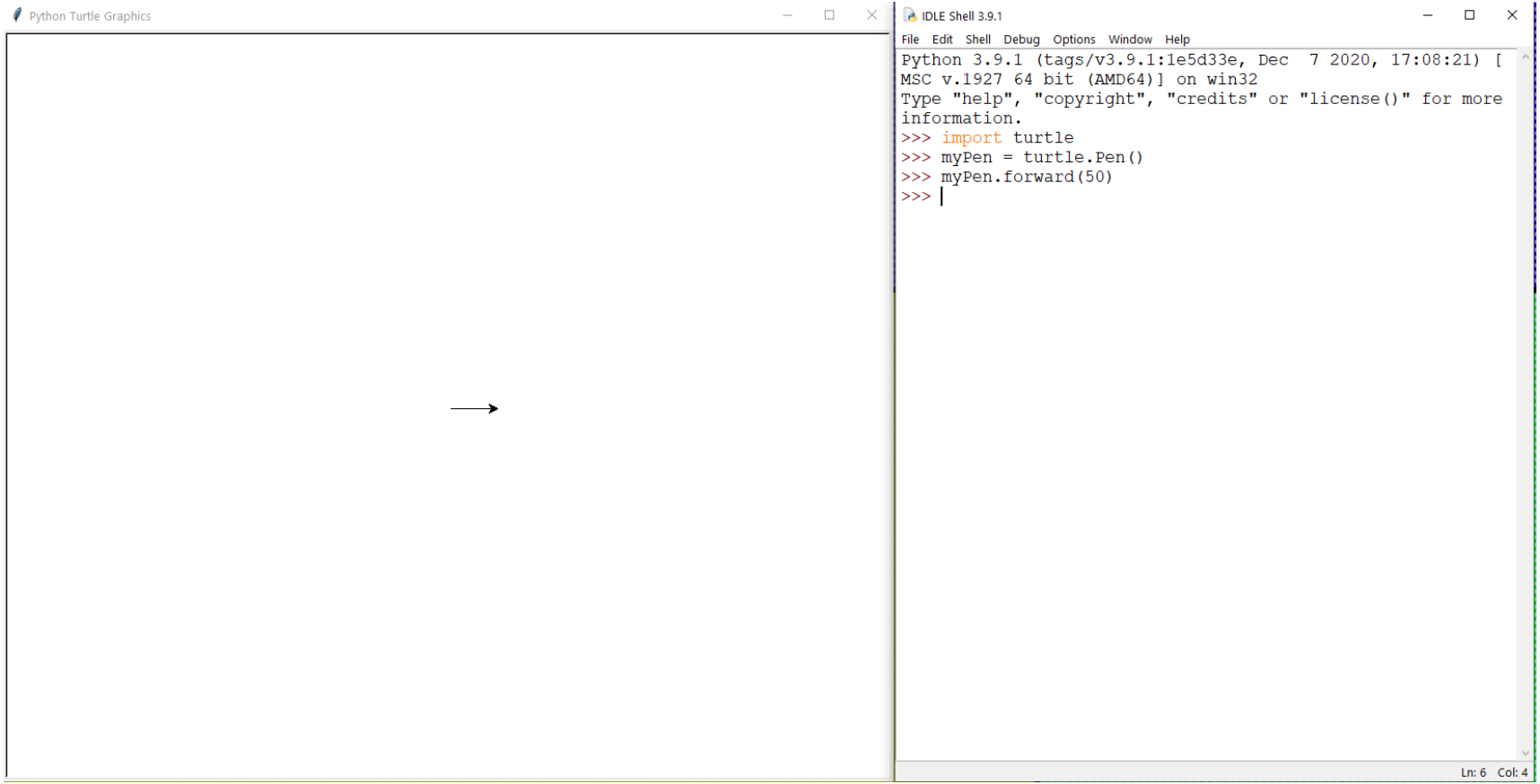
Turtle Drawing Commands

pendown()	Put the pen down—drawing when moving.
penup()	Pull the pen up—no drawing when moving.
pensize(*width*)	Set the line thickness to *width*, a positive int.
circle(*radius*, *extent=None*, *steps=None*)	Draw a circle (see below).
dot(*size=None*, color)	Draw a filled dot.
stamp()	Stamp a copy of the turtle at the present position. Returns a `stamp_id`.
clearstamp(*stamp_id*)	Remove stamp with `stamp_id`.
clear()	Clear the screen. Leave the turtle position and orientation unchanged.
reset()	Clear screen. Reset turtle to initial configuration (at (0,0) facing east).

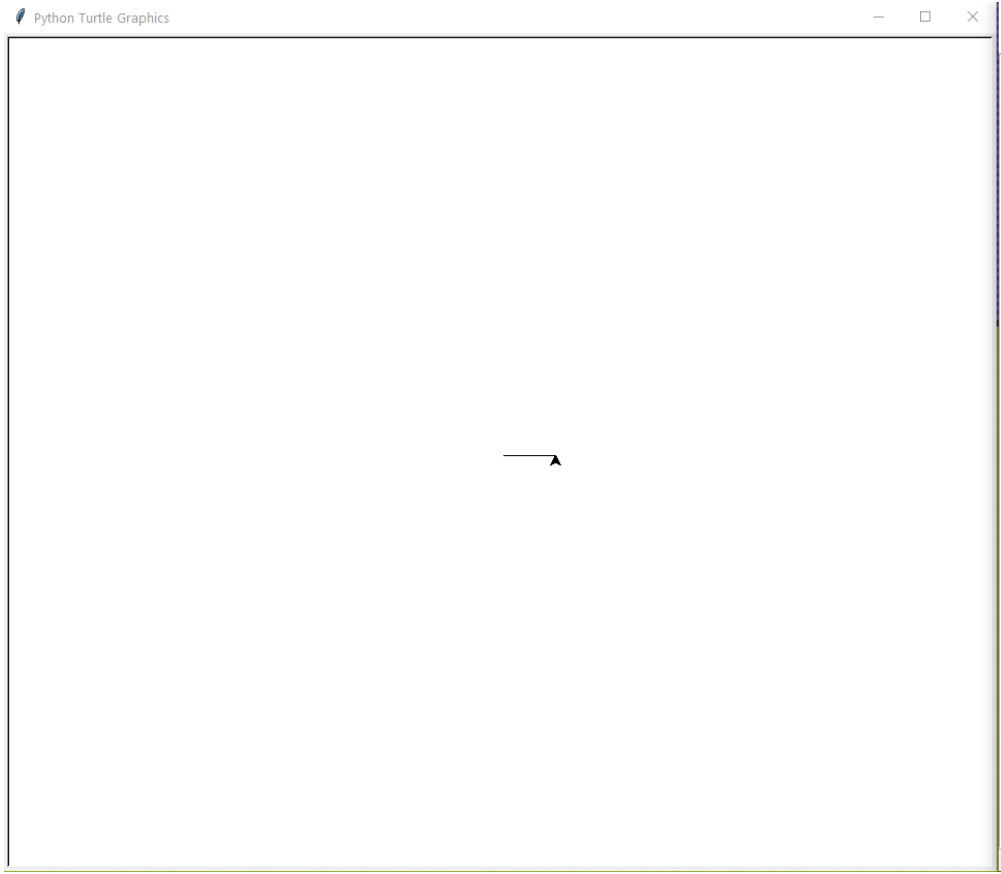
Turtle Graphics

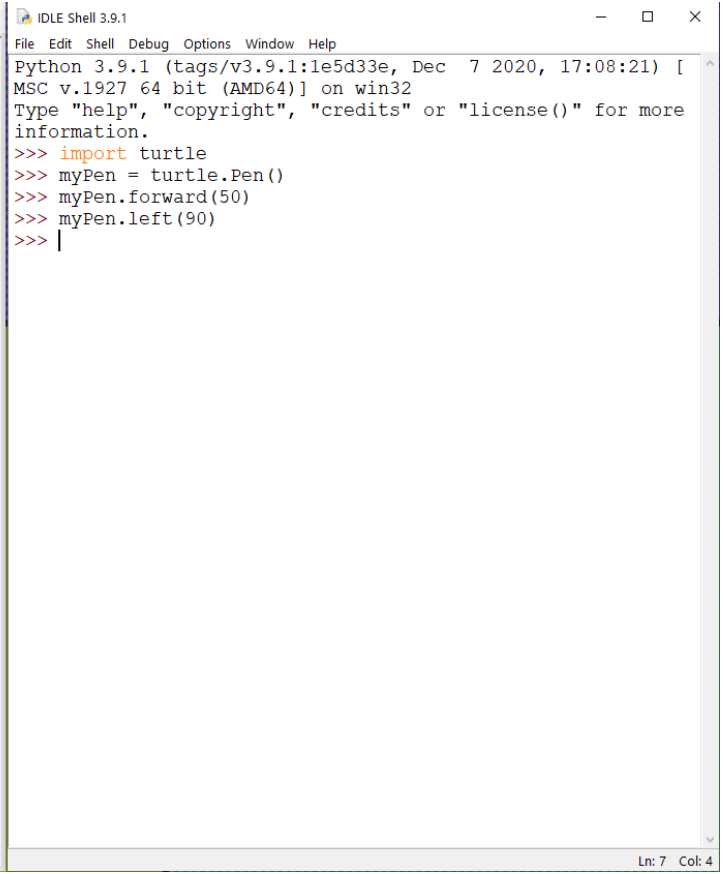


Turtle Graphics

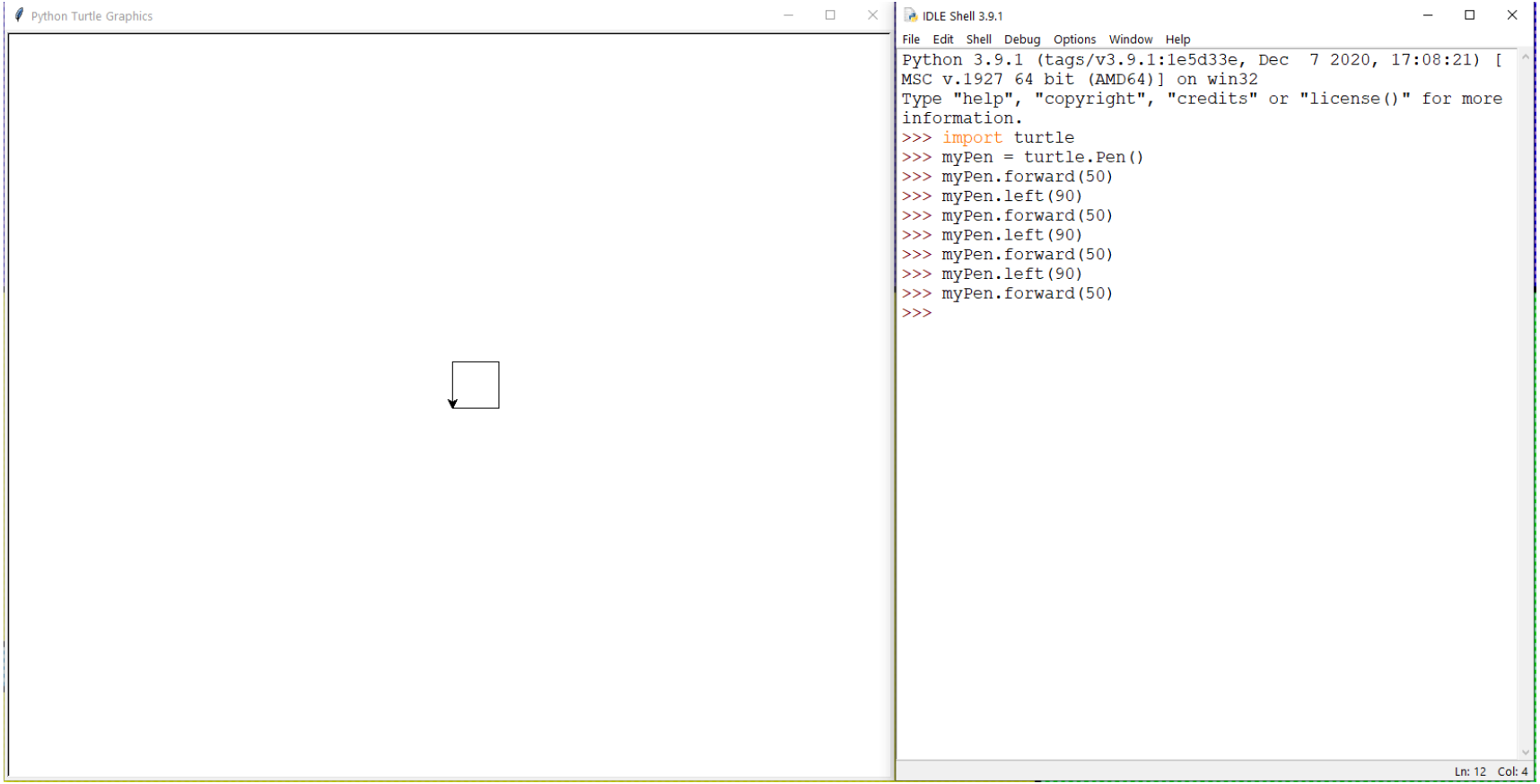


Turtle Graphics

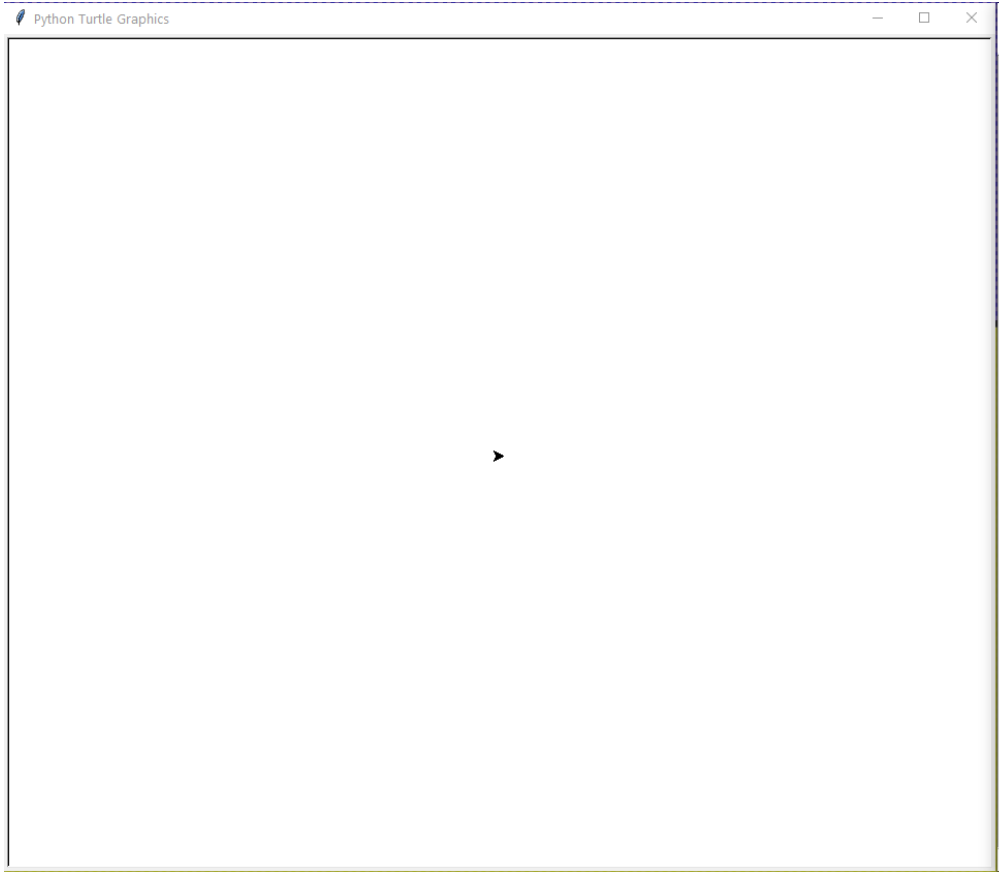


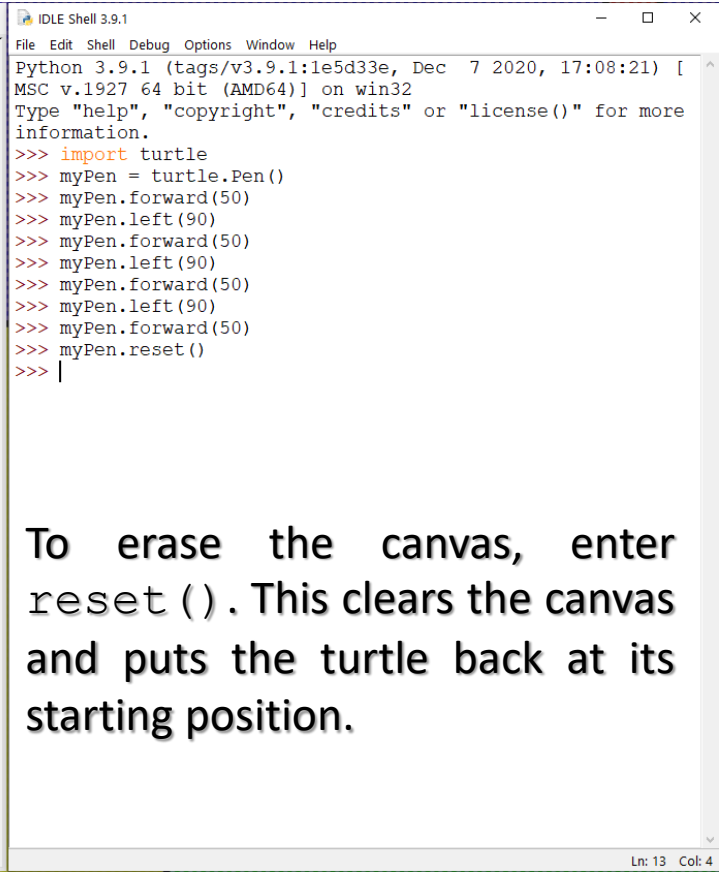


Turtle Graphics



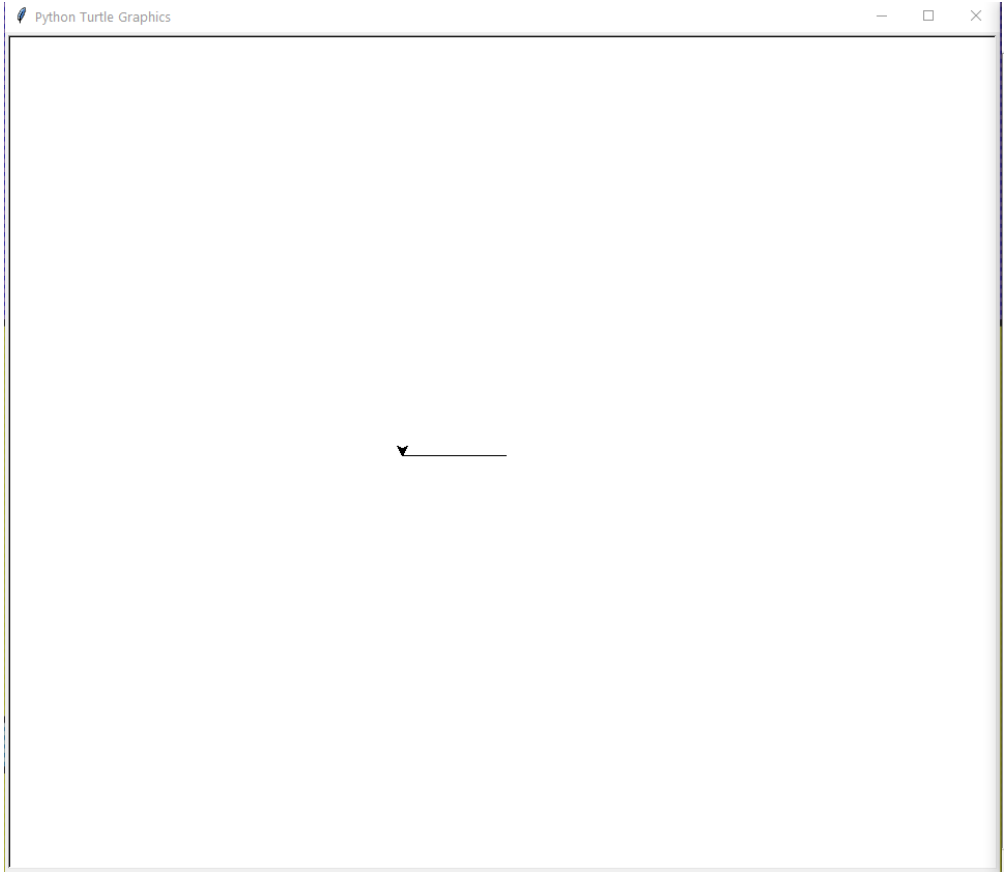
Turtle Graphics

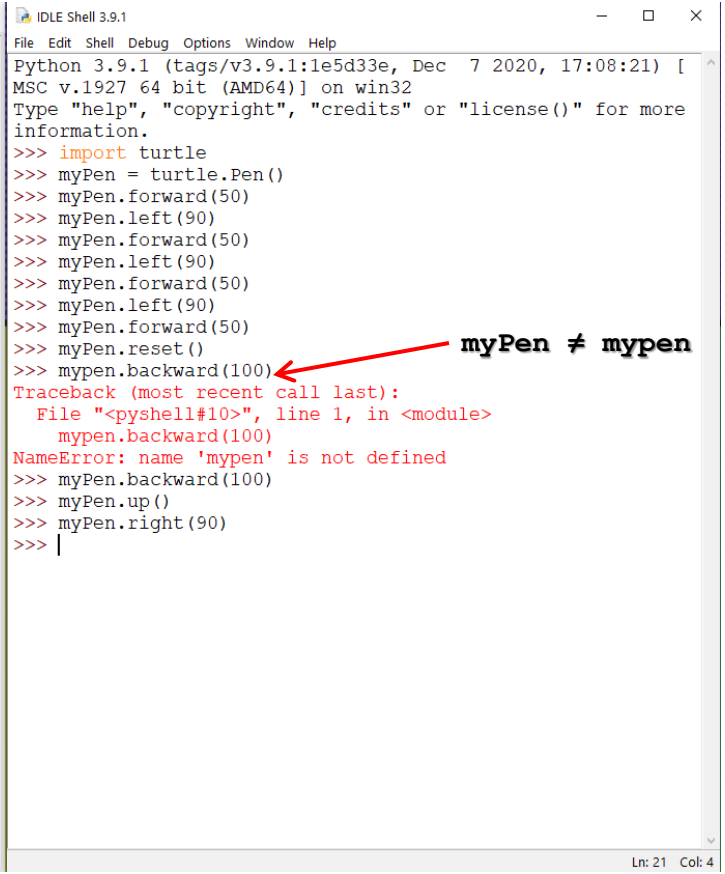




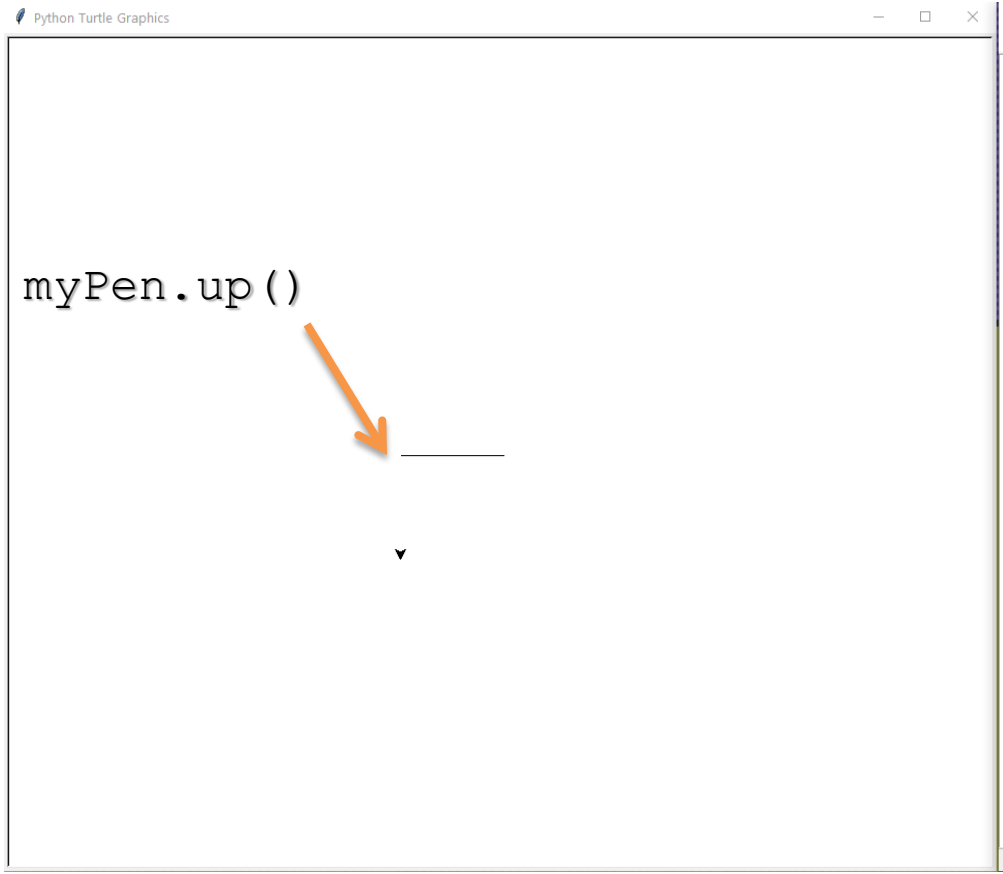
To erase the canvas, enter `reset()`. This clears the canvas and puts the turtle back at its starting position.

Turtle Graphics





Turtle Graphics



IDLE Shell 3.9.1

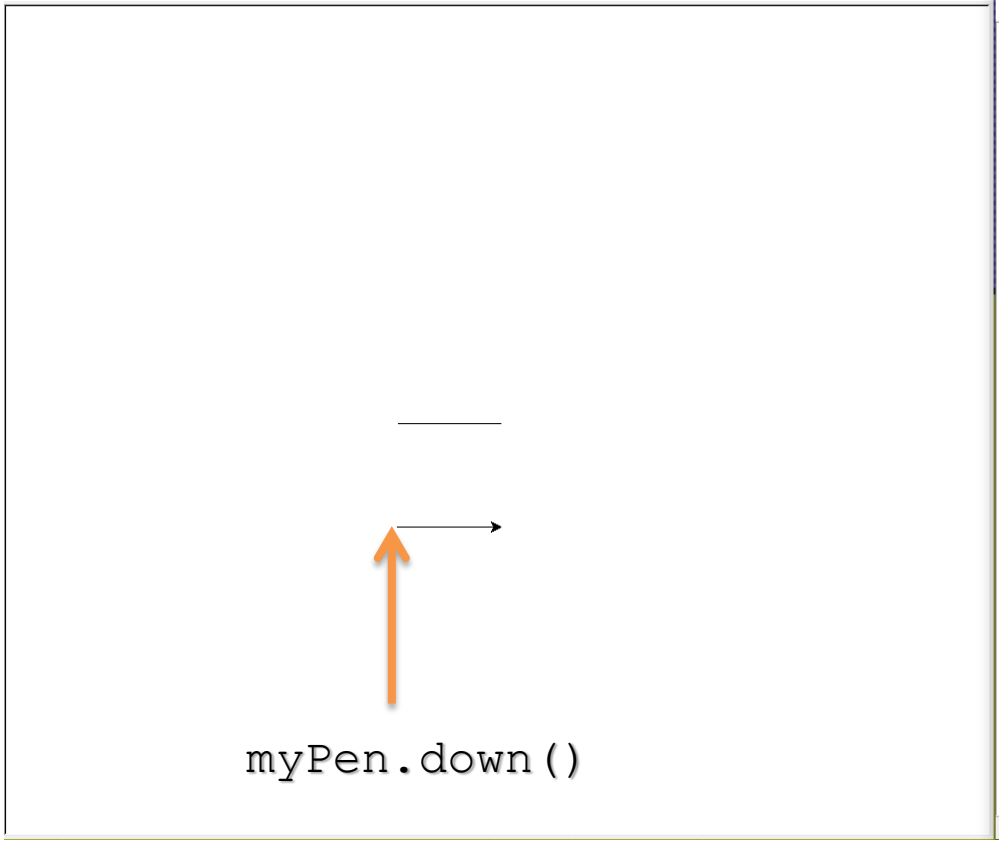
File Edit Shell Debug Options Window Help

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [
MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>> import turtle
>>> myPen = turtle.Pen()
>>> myPen.forward(50)
>>> myPen.left(90)
>>> myPen.forward(50)
>>> myPen.left(90)
>>> myPen.forward(50)
>>> myPen.left(90)
>>> myPen.forward(50)
>>> myPen.reset()
>>> mypen.backward(100)
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    mypen.backward(100)
NameError: name 'mypen' is not defined
>>> myPen.backward(100)
>>> myPen.up()
>>> myPen.right(90)
>>> myPen.forward(100)
>>>
```

Ln: 22 Col: 4

Turtle Graphics

Python Turtle Graphics



IDLE Shell 3.9.1

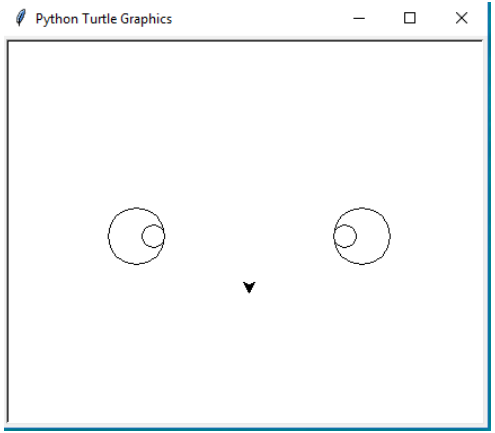
File Edit Shell Debug Options Window Help

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [
MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>> import turtle
>>> myPen = turtle.Pen()
>>> myPen.forward(50)
>>> myPen.left(90)
>>> myPen.forward(50)
>>> myPen.left(90)
>>> myPen.forward(50)
>>> myPen.left(90)
>>> myPen.forward(50)
>>> myPen.reset()
>>> mypen.backward(100)
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    mypen.backward(100)
NameError: name 'mypen' is not defined
>>> myPen.backward(100)
>>> myPen.up()
>>> myPen.right(90)
>>> myPen.forward(100)
>>> myPen.down()
>>> myPen.left(90)
>>> myPen.forward(100)
>>>
```

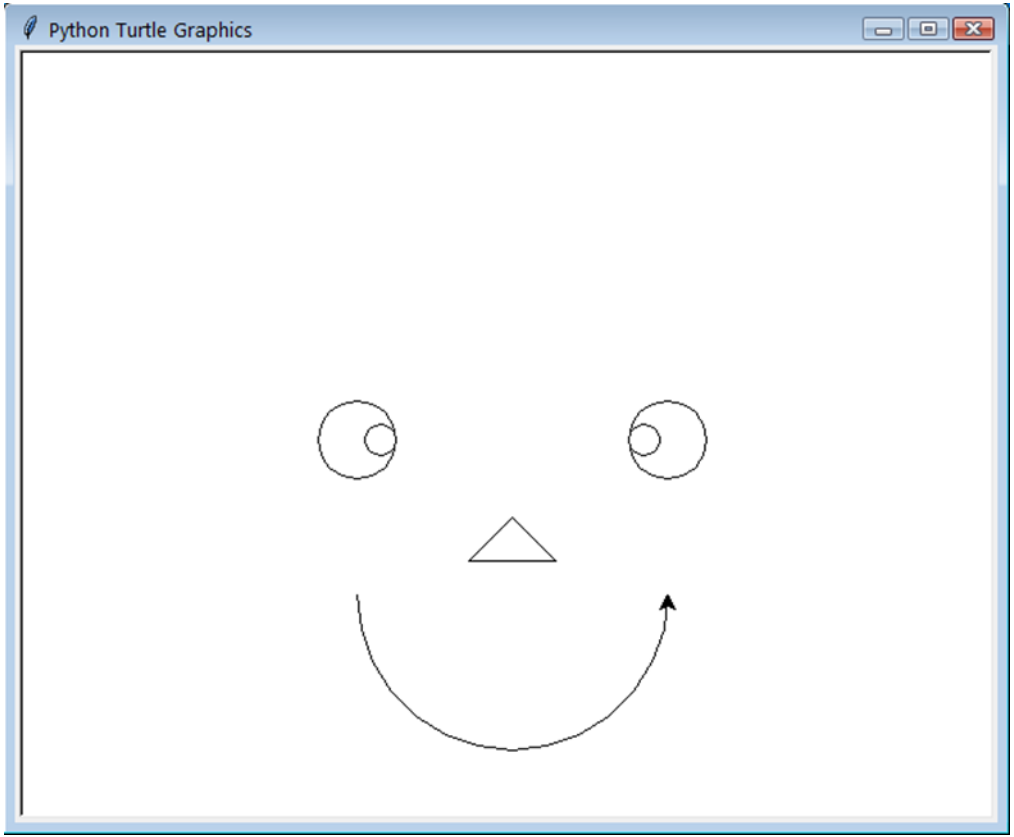
Ln: 25 Col: 4

Turtle Graphics

```
# draw a smiley face
from turtle import *

speed(10)    # draw fast !
penup()      # right s ide of face
forward(75)
pendown()    # draw an eye
right(90)
circle(25)
circle(10)
penup()      # left side of face
right(90)
forward(150)
pendown()    # draw an eye
right(90)
circle(25)
circle(10)
penup()      # center and down
right(90)
forward(75)
right(90)
forward(50)
```

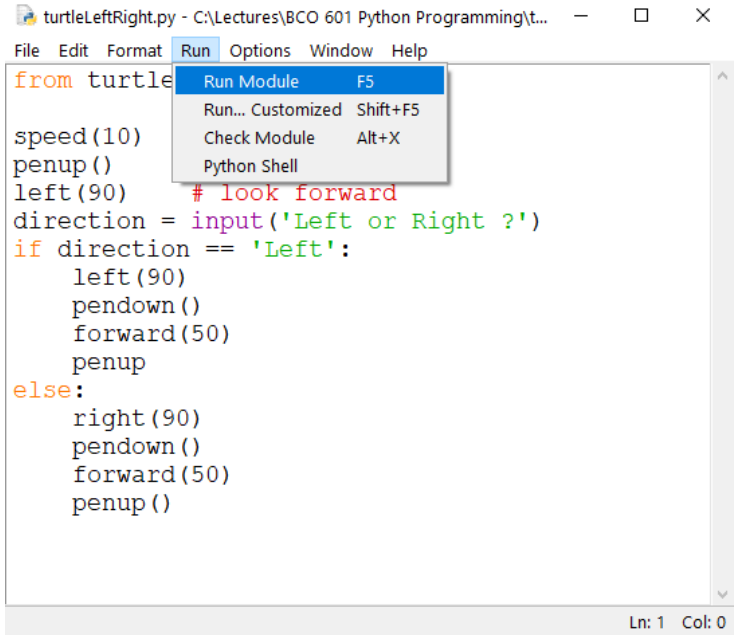


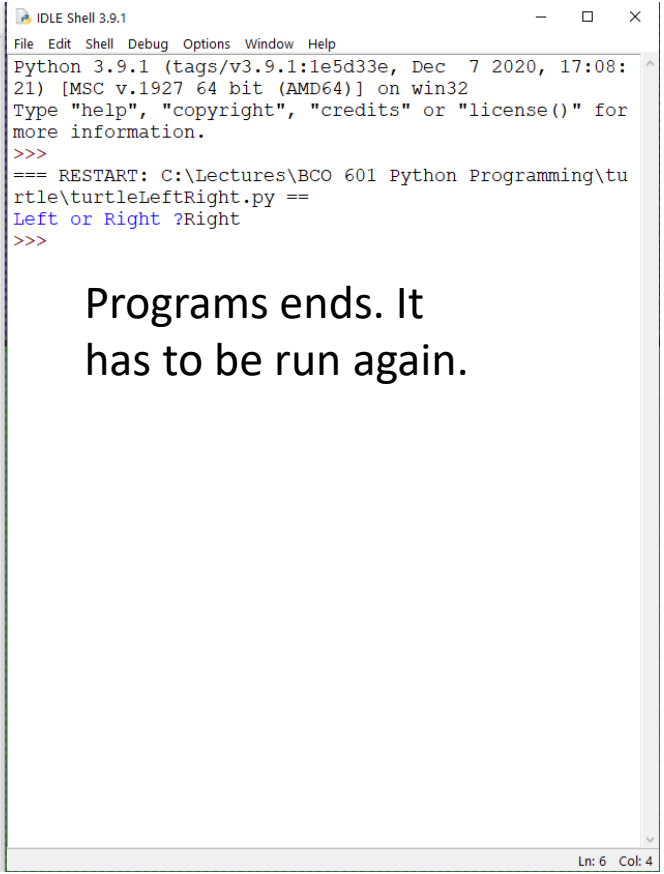
Homework 1 : try to draw a smiley face



if/else in Turtle Graphics

```
from turtle import *

speed(10)     # draw fast !
penup()
left(90)      # look forward
direction = input('Left or Right ?')
if direction == 'Left':
     left(90)
     pendown()
     forward(50)
     penup
else:
     right(90)
     pendown()
     forward(50)
     penup()
```





Programs ends. It has to be run again.

if/else in Turtle Graphics

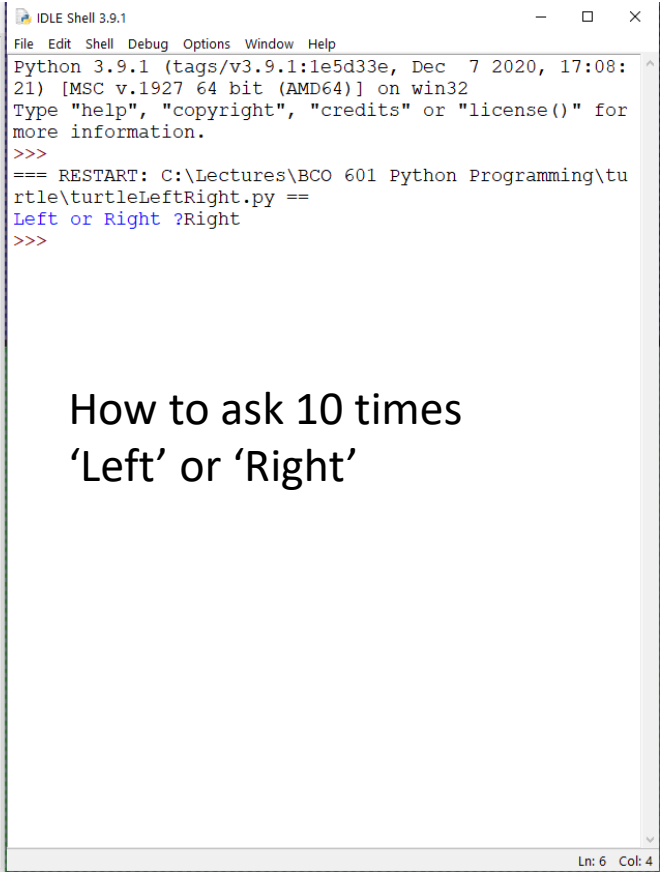
```
from turtle import *

speed(10)    # draw fast !
penup()
left(90)     # look forward
direction = input('[L]eft [R]ight [U]p [D]own ? ')
if direction == 'Left' or direction == 'L':' :

   …
   …
```

```
today = 'Saturday'

if today == 'Sunday' or today == 'Saturday':
        print('Today is off. Rest at home.')
else:
        print('Go to work.')
```

OR Truth Table

Inputs		Output
A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

```
today == 'Sunday'
today == 'Saturday'
```



How to ask 10 times ‘Left’ or ‘Right’

The `in` Operator

`in` operator Returns `True` if a sequence with the specified value is present in the object

```
>>> 'r' in 'serdar'
True
>>> isim = "serdar"
>>> 'e' in isim
True
```

`not in` operator Returns `True` if a sequence with the specified value is NOT present in the object

```
>>> 'r' not in 'serdar'
False
>>> 't' not in 'serdar'
True
```

The `in` Operator

```
>>> yas = 58
>>> 5 in yas
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    5 in yas
TypeError: argument of type 'int' is not iterable

>>> '5' in str(yas)
True
```

An **iterator** is an object that contains **a countable number** of values.

An **iterator** is an object that can be iterated upon, meaning that you can traverse through all the values.

Iterator vs Iterable

```
>>> myname = "Serdar"
>>> my_iterator = iter(myname)
>>> print(next(my_iterator))
S
>>> print(next(my_iterator))
e
>>> print(next(my_iterator))
r
>>> print(next(my_iterator))
d
>>> print(next(my_iterator))
a
>>> print(next(my_iterator))
r
>>> print(next(my_iterator))
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in
  <module>
    print(next(my_iterator))
StopIteration
```

Lists, tuples, dictionaries, sets and strings are all iterable objects. They are iterable containers which you can get an iterator from. All these objects have a `iter()` method which is used to get an iterator operator

The For Loop

for loop: Repeats a set of statements over a group of values.

- **Syntax:**

```
for variableName in groupOfValues:
    statements
```

indent

We indent the statements to be repeated with tabs or spaces.

`variableName` gives a name to each value, so you can refer to it in the statements.

`groupOfValues` can be a range of integers, specified with the `range` function.

The For Loop

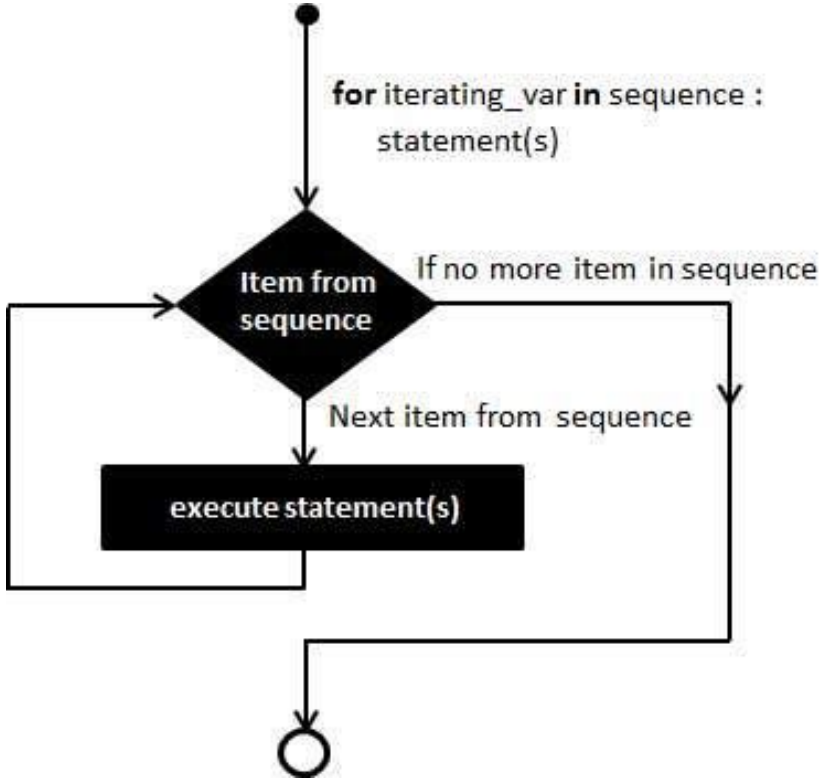
- **Example:**

```
for x in range(6):
    print (x)
```

Output:

```
0
1
2
3
4
5
```

! There is not number 6



The For Loop

– **Example:**

range()?

```
for harfler in 'Serdar':
        print('Harf : ', harfler)
```

```
Harf :  S
Harf :  e
Harf :  r
Harf :  d
Harf :  a
Harf :  r
```

range

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

```
>>> mynumber = range(3)
>>> my_iterator = iter(mynumber)
>>> print(next(my_iterator))
0
>>> print(next(my_iterator))
1
>>> print(next(my_iterator))
2
>>> print(next(my_iterator))
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    print(next(my_iterator))
StopIteration
```

The For Loop

– **Example:**

```
for x in range(1, 6):
    print (x, "squared is", x**2)
```

```
Output:
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

The For Loop

Syntax

```
for variable in range(...) :
    statements
```

This variable is set, at the beginning of each iteration, to the next integer in the sequence generated by the range function.

The range function generates a sequence of integers over which the loop iterates.

```
for i in range(5) :
    print(i)   # Prints 0, 1, 2, 3, 4
```

With one argument, the sequence starts at 0. The argument is the first value NOT included in the sequence.

```
for i in range(1, 5) :
    print(i)   # Prints 1, 2, 3, 4
```

With two arguments, the sequence starts with the first argument.

```
for i in range(1, 11, 2) :
    print(i)   # Prints 1, 3, 5, 7, 9
```

With three arguments, the third argument is the step value.

The For Loop
range

range(start, stop) - the integers between start (inclusive) and stop (exclusive)

It can also accept a third value specifying the change between values.

- **range(start, stop, step) - the integers between start (inclusive) and stop (exclusive) by step**

Example:

```
for x in range(5, 0, -1):
    print(x)
print ("Blastoff!")
Output:
5
4
3
2
1
Blastoff!
```

The For Loop range

Example:

```
>>> for i in range(1,7):
        print (i, i**2, i**3, i**4)
  1 1 1 1
  2 4 8 16
  3 9 27 81
  4 16 64 256
  5 25 125 625
  6 36 216 1296
>>> for x in range(0, 5):
    print('hello %s' % x)
hello 0
hello 1
hello 2
hello 3
hello 4
```

The For Loop
Cumulative Loops

Some loops incrementally compute a value that is initialized outside the loop. This is sometimes called a cumulative sum.

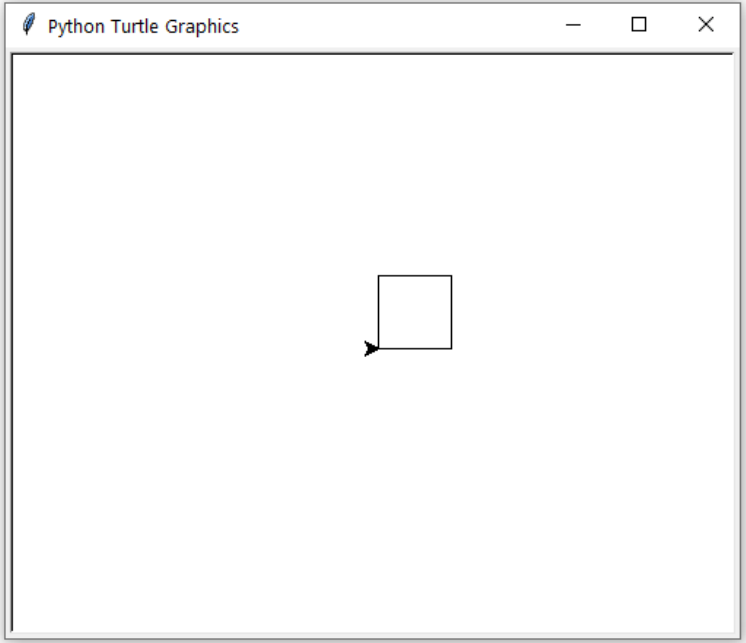
```
sum = 0
for i in range(1, 11):
    sum = sum + (i * *2)
print ("sum of first 10 squares is", sum)

Output:
sum of first 10 squares is 385
```

- **Homework Exercise: Write a Python program that computes the factorial of an integer. Factorial > 5! = 5 * 4 * 3 * 2 * 1 = 120**

The For Loop

for loop in Turtle Graphics

```
from turtle import *

speed(10)    # draw fast !
for i in range(4):
   forward(50)
   left(90)

Variable 'i' is not used, so

for _ in range(4):
   forward(50)
   left(90)
```



The For Loop

for loop in Turtle Graphics

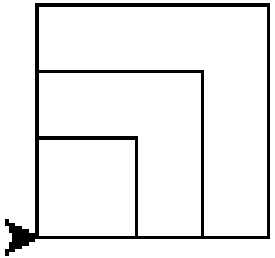
```
from turtle import *

speed(10)    # draw fast !
penup()
left(90)     # look forward
for i in range(10):
    direction = input('[L]eft [R]ight [U]p [D]own ? ')
    if direction == 'left' or direction == 'l':
```

The For Loop

```
from turtle import *

shape('arrow') # convert to arrow
speed(10)      # draw fast !
left(90)       # look forward
for i in range(10):
    direction = input('[L]eft [R]ight [F]orward [B]ackward ? ')
    if direction.lower() == 'left' or direction.lower() == 'l':
        left(90)
        forward(100)
    elif direction.lower() == 'right' or direction.lower() == 'r':
        left(90)
        forward(100)
    elif direction.lower() == 'forward' or direction.lower() == 'f':
        forward(100)
    elif direction.lower() == 'backward' or direction.lower() == 'b':
        backward(100)
    else:
        print('Wrong key: you can only choose [L]eft [R]ight [F]orward [B]ackward')
```

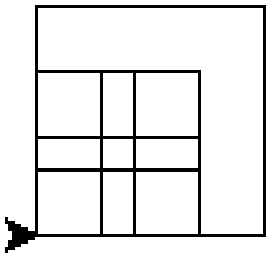
The For Loop

```
from turtle import *

speed(10)    # draw fast !
for _ in range(4):
   forward(70)
   left(90)

for _ in range(4):
   forward(50)
   left(90)

for _ in range(4):
   forward(30)
   left(90)
```



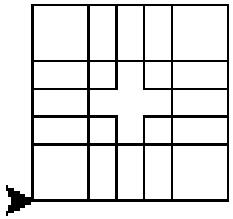
The For Loop

```
from turtle import *

speed(10)    # draw fast !
for _ in range(4):
   forward(70)
   left(90)

for _ in range(4):
   forward(50)
   left(90)

   for _ in range(4):
       forward(30)
       left(90)
```



The For Loop

```
from turtle import *

speed(10)    # draw fast !
for _ in range(4):
    forward(70)
    left(90)

    for _ in range(4):
        forward(50)
        left(90)

    for _ in range(4):
        forward(30)
        left(90)
```



The For Loop
pass

It is used when a statement is required syntactically but you do not want any command or code to execute. The `pass` statement is a null operation; nothing happens when it executes.

```
for letter in 'Python':
   if letter == 'h':
       pass
       print ('This is pass block')
   print('CurrentLetter:', letter)
print ('Good bye!')
```

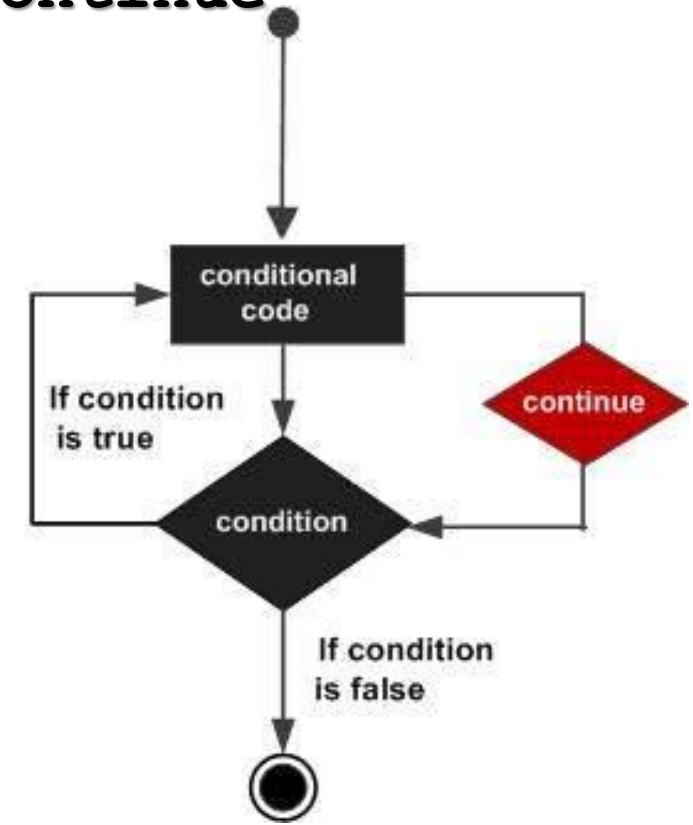
The For Loop

continue

It returns the control to the beginning of the while loop

The `continue` statement can be used in both `while` and `for` loops.

```
for letter in 'Python':
    if letter == 'h':
        continue
    print('CurrentLetter:', letter)
```

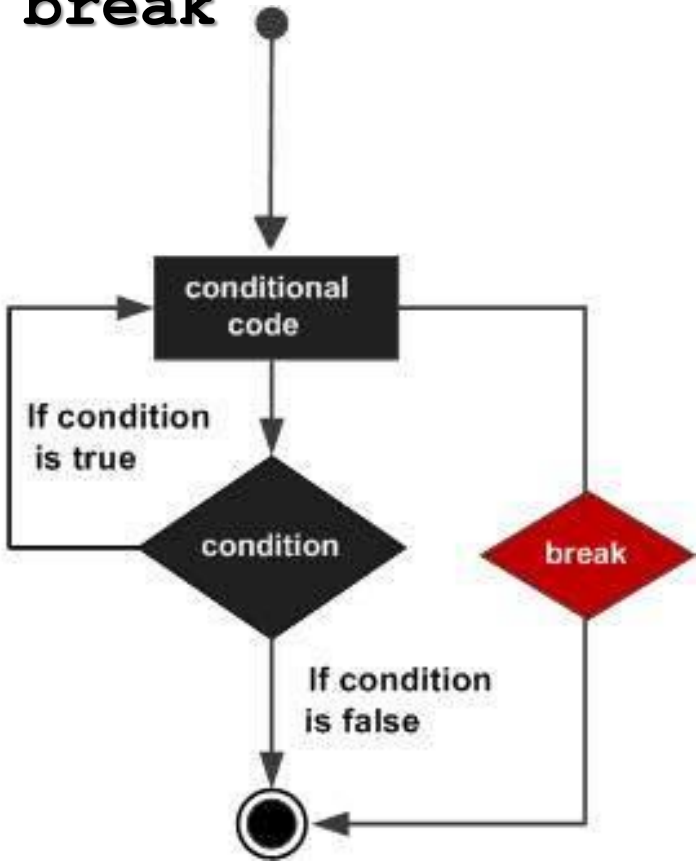


The For Loop `break`

It terminates the current loop and resumes execution at the next statement.

The `break` statement can be used in both `while` and `for` loops.

```
for letter in 'Python'
   if letter == 'h':
      break
   print('CurrentLetter:', letter)
```



pass, continue, break

```
for letter in 'Python':
    if letter == 'h':
        pass
        print('This is pass block')
    print('CurrentLetter:', letter)
print('Good bye!')
```

```
CurrentLetter: P
CurrentLetter: y
CurrentLetter: t
This is pass block
CurrentLetter: h
CurrentLetter: o
CurrentLetter: n
Good bye!
```

```
for letter in 'Python':
    if letter == 'h':
        continue
        print('This is continue block')
    print('CurrentLetter:', letter)
print('Good bye!')
```

```
CurrentLetter: P
CurrentLetter: y
CurrentLetter: t
CurrentLetter: o
CurrentLetter: n
Good bye!
```

```
for letter in 'Python':
    if letter == 'h':
        break
        print('This is break block')
    print('CurrentLetter:', letter)
print('Good bye!')
```

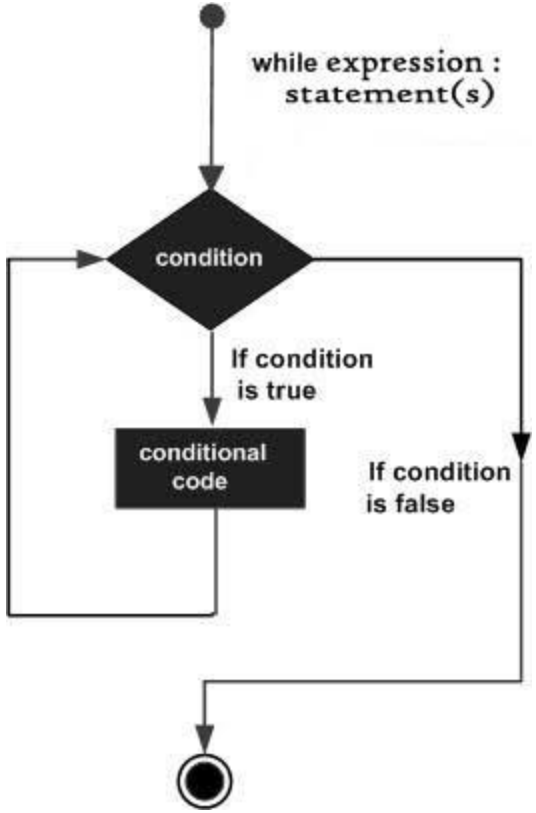
```
CurrentLetter: P
CurrentLetter: y
CurrentLetter: t
Good bye!
```

The While Loop

A `while` loop statement in Python programming language repeatedly executes a target statement as long as a given condition is `true`.

The syntax of a while loop in Python programming language is

```
while expression:
    statement(s)
```



The While Loop

As long as the condition is true, the while statement will execute the action

Example:

```
>>> x = 1
>>> while x < 4:                    # as long as x < 4...
        print( x**2)                # print the square of x
        x = x + 1                   # increment x by +1

1 # only the squares of 1, 2, and 3 are printed, because
4 # once x = 4, the condition is false
9
```

The While Loop

Pitfall to avoid:

While statements are intended to be used with changing conditions. If the condition in a while statement does not change, the program will be stuck in an infinite loop until the user hits ctrl-C.

Example:

```
>>> x = 1
>>> while x == 1:
      print ('Hello world')
```

Since x does not change, Python will continue to print "Hello world" until interrupted

The While Loop

Using else Statement

Python supports to have an `else` statement associated with a loop statement. The else statement is executed when the condition becomes false.

```
count = 0
while count < 5:
   print (count, " is  less than 5")
   count = count + 1
else:
   print(count, " is not less than 5")
```

```
0  is  less than 5
1  is  less than 5
2  is  less than 5
3  is  less than 5
4  is  less than 5
5  is not less than 5
```

How to quit

```
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program."

message = ""

while message != 'quit':
    message = input(prompt)
    print(message)
```

```
Tell me something, and I will repeat it back to
you:
Enter 'quit' to end the program.


Tell me something, and I will repeat it back to
you:
Enter 'quit' to end the program. serdar
serdar

Tell me something, and I will repeat it back to
you:
Enter 'quit' to end the program. quit
quit
```

How to quit: using a flag

```
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program."

message = ""

active = True
while active:
    message = input(prompt)

    if message == 'quit':
        active = False
    else:
        print(message)
```

How to quit: using break

```
prompt = "\nPlease enter the name of a city you have visited:"
prompt += "\n(Enter 'quit' to end the program.)"

message = ""

while True :

    message = input(prompt)

    if message == 'quit':
        break
    else:
        print(f"I'd love to go to {message.title()}!")
```

```
Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) London
I'd love to go to London!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) paris
I'd love to go to Paris!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) istanbul
I'd love to go to Istanbul!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) quit
```

Python '!=' Is Not 'is not':

```
while message != 'quit':
while message is not 'quit':
```

The **!=** operator compares only the value of the objects being compared.

The "**is not**" operator compares if the objects are pointing to the same memory location or not.

It returns True if the value of both the objects are different and False otherwise.

String Formatting

In its simplest form, it is possible to use a string's format method to insert objects into it.

Example:

```
>>>'{} plus {} equals {} '.format(2, 3, 'five ')
'2 plus 3 equals five '
>>>'{1} plus {0} equals {2} '.format(2, 3, 'five ')
'3 plus 2 equals five '
>>>'{n_1} plus {n_2} equals {ans}'.format(n_1=2, n_2=3, ans='five ')
'2 plus 3 equals five '
>>>'{0} plus {0} equals {1} '.format(2, 2+2)
'2 plus 2 equals 4'
```

String Formatting

```
>>> '=== {0: <12} === '.format('Python ')
'=== Python       === '
>>> '=== {0: >12} === '.format('Python ')
'===       Python === '
>>> '=== {0:^12} === '.format('Python ')
'===    Python    === `
>>> '=== {0: -^12} === '.format ('Python ')
'=== ---Python --- === `
>>> a = 254
>> 'a ={0:5d}'.format(a)       # decimal
'a = 254 '
>>> 'a = {0:10 b}'.format(a)  # binary
'a = 11111110 '
>>> 'a = {0:5x}'.format(a)    # hex (lower -case)
'a = fe'
```

String Formatting

Since version 3.6, Python has supported a further way of interpolating values into strings: a string literal denoted with a `f` before the quotes can evaluate expressions placed within braces, including references to variables, function calls and comparisons.

Example:

```
>>> h = 6.62607015e-34
>>> unit = 'J.s'
```

instead of using the format function:

```
>>> 'h = {h:.3e} {unit}'.format(h=h, unit = unit )
'h = 6.626e-34 J.s'
```

one can simply write:

```
>>> f'h = {h:.3e} { h_units }'
'h = 6.626e-34 J.s'
```

String Formatting

```
>>> name = 'Elizabeth '
>>> f'The name {name} has {len(name)} letters and {name.lower().count("e")}
"e"s.'
'The name Elizabeth has 9 letters and 2 "e"s.'
```

or even:

```
>>> letter = 'k'
>>> f'{name} has {len(name)} letters and {name.lower().count(letter)}
"{letter}"s.'
'Elizabeth has 9 letters and 0 "k"s.'
```

f-strings are evaluated once, at runtime, it is not possible to define a reuseable "template":

```
>>> radius = 2.5
>>> s = f'The radius is { radius } m.'
>>> print(s)
The radius is 2.5 m.
>>> radius = 10.3
>>> print(s)
The radius is 2.5 m.
```

Exercises

- **Please write a simple averaging code**

Sample output:

```
Lütfen 10 adet sayı giriniz
1.sayıyı giriniz : 1
2.sayıyı giriniz : 2
3.sayıyı giriniz : 3
4.sayıyı giriniz : 4
5.sayıyı giriniz : 5
6.sayıyı giriniz : 6
7.sayıyı giriniz : 7
8.sayıyı giriniz : 8
9.sayıyı giriniz : 9
10.sayıyı giriniz : 10
10 sayının toplamı 55 ve ortalaması : 5.5
```

Homework 2

Number guessing game in Python. Inputs range, let's say from 1 to 100.

```
    You've only  7  chances to guess the integer!

Guess a number:- 50
You guessed too small!
Guess a number:- 75
You Guessed too high!
Guess a number:- 62
You Guessed too high!
Guess a number:- 56
You Guessed too high!
Guess a number:- 53
You guessed too small!
Guess a number:- 54
Congratulations you did it in  6  try
```

```
print ("you have only 7 chances to guess the integer!")

hidden_number = 23

your_guess = int(input ('Guess a number: '))
if your_guess == hidden_number:
    print ("Congratulations you did it in 1 try")
elif your_guess < hidden_number:
    print("You guessed too small")
else:
    print("You guessed too high")
```

Homework 3

Guessing Numbers : The problem is to guess what number a computer has in mind. You will write a program that randomly generates an integer between 0 and 100, inclusive. For each user input, the program reports whether it is too low or too high, so the user can choose the next input intelligently. Here is a sample run:

```
Guess a magic number between 0 and 100
Enter your guess: 50
Your guess is too high
Enter your guess: 25
Your guess is too low
Enter your guess: 42
Your guess is too high
Enter your guess: 39
Yes, the number is 39
```

```
import random
# Integer from 1 to 100, endpoints included
number = random.randint(0, 100)
```

Homework 4

- **Write a program that reads a word, and prints the number of letters in the word, the number of vowels in the word, and the percentage of vowels.**

```
Enter a word: sequoia
Letters: 7
Vowels: 5
Percentage of vowels: 71.42
```

Exercises

- Write a Python script that calculates the average value of given numbers. Please write the same code with - (minus) number enterence

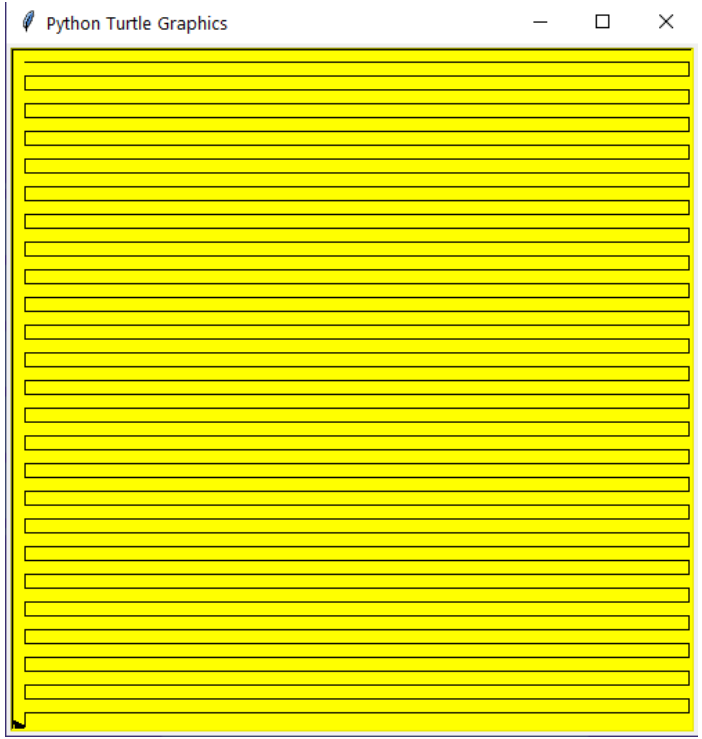
This is Output of the script

```
enter your number: 12
enter your number: -1
average is 12.0

enter your number: 12
enter your number: 9
enter your number: 18
enter your number: -1
average is 13.0
```

Exercises

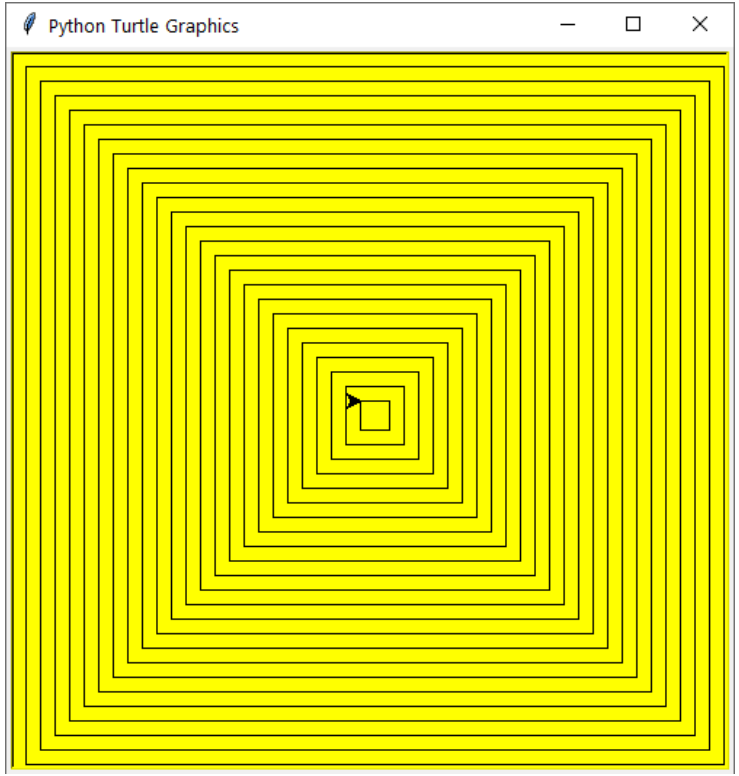
```
import turtle

# set up stage or canvas
turtle.setup(500, 500)
# set the background color
turtle.bgcolor("yellow")
turtle.penup()
turtle.goto(-240, 240)
turtle.pendown()
turtle.forward(480)
turtle.right(90)
turtle.forward(10)
turtle.right(90)
turtle.forward(480)
.
.
.
.
```

- **Write a program that draws the shape below.**



Exercises

- **Write a program that draws the shape below.**



Exercises

The `random.randint()` method returns an integer number selected element from the specified range.

Note: This method is an alias for `randrange(start, stop+1)`.

Syntax

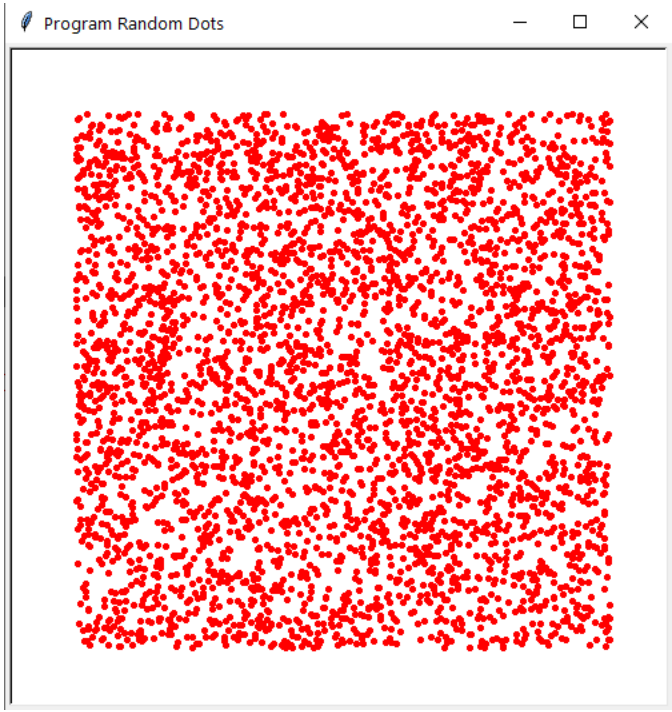
```
random.randint(start, stop)
```

Parameter Values

Parameter	Description
start	Required. An integer specifying at which position to start.
stop	Required. An integer specifying at which position to end.

Exercises

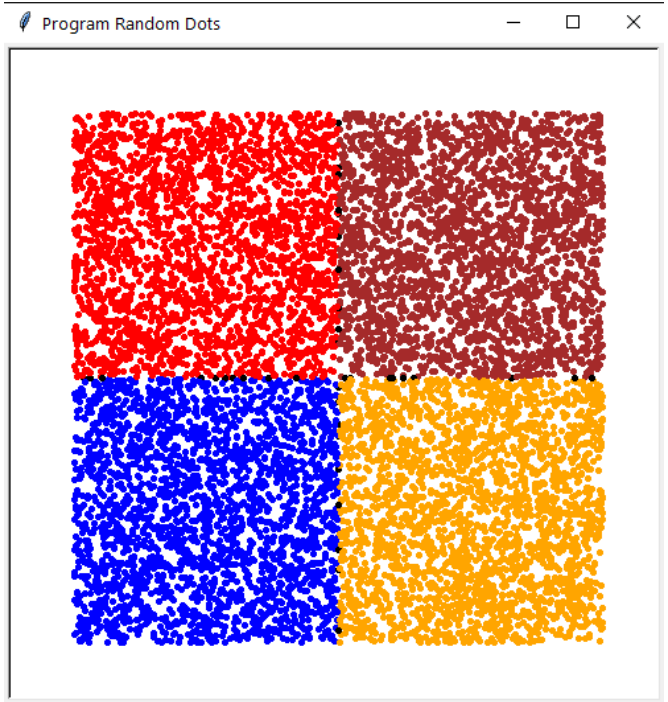
```
import turtle
import random
turtle.title("Program Random Dots")
turtle.setup(500, 500)

for _ in range(100): # make it 10000
    # choose a random spot
    xpos = random.randint(-200,200)
    ypos = random.randint(-200,200)

    # goto this spot
    turtle.penup()
    turtle.goto(xpos, ypos)
    turtle.pendown()
    turtle.dot(5, "red")
```



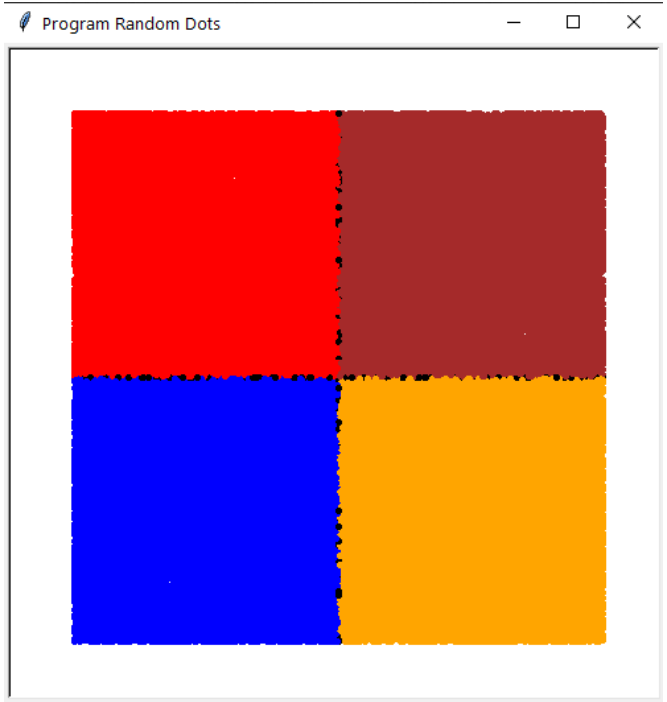
Exercises

- **Write a program that draws the shape below.**

```
for _ in range(10000):
```

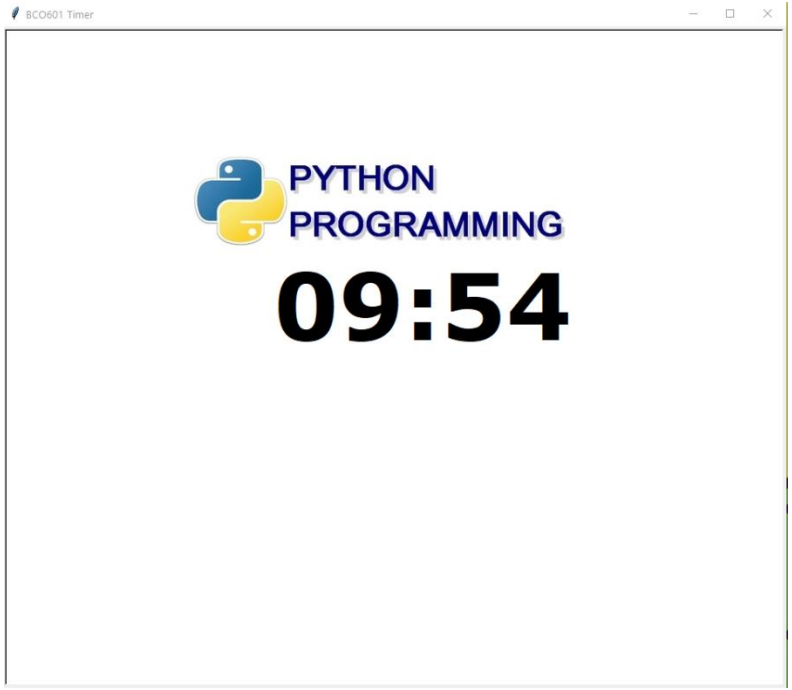



```
for _ in range(100000):
```



Turtle Graphics

online learning during the COVID-19 pandemic



Turtle Graphics

```
import turtle
import time

# define the countdown function
def countdown(t):

    while t:
        mins, secs = divmod(t, 60)
        timer = '{:02d}:{:02d}'.format(mins, secs)
        turtle.backward(150)
        turtle.write(timer, font=("Verdana", 85, "bold"))
        time.sleep(1)
        t -= 1
        turtle.reset()
    turtle.backward(400)
    turtle.write('The lecture is starting', font=("Verdana", 55, "normal"))

turtle.title('BCO601 Timer')
turtle.hideturtle()
turtle.bgpic("counterBack.png")
# input time in seconds
t = input("Enter the time in seconds: ")
screen = turtle.Screen()
screen.tracer(False)
# function call
countdown(int(t))
```