Mr Baumgarten
3 November 2017

python™ LANGUAGE REFERENCE

1. PRINT AND INPUT

Print a string

```
print("hello world!")
```

Set and print a string variable

```
name = "Jon Snow"
print("Hello ", name )
```

Prompt the user for string input

```
name = input("What is your name?")
print("Hello ", name )
```

Prompt the user for an integer input

```
num = int( input("Type a number:") )
print( "You typed:", num )
```

OVERRIDE DEFAULT BEHAVIOUR OF PRINT

- By default, print() will insert a space between each parameter it prints, and end with a new line break.

Changing the separator to join multiple parameters together

```
print("Hello", "world!", sep="")          ## helloworld!
print("Hello", "world!", sep="---")       ## hello---world!
```

Changing the end of line behavior

```
print("Hello", "world!", end="")          ## hello world! ... without a line break
```

2. VARIABLES & NUMBERS

COMMON TYPES OF VARIABLES

- Integers — Whole numbers eg: 0, 13, -5
- Floating point numbers — Numbers with decimal/fractional parts eg 3.1415
- Strings — Alpha/numeric text enclosed with quotes eg: “Wazzup!”
- Booleans — True or False

NUMERIC OPERATIONS

- Addition
- Subtraction
- Multiplication
- Floating point division
- Integer division
- Modulus (remainder)
- Exponent

```
a = 10
b = 4
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a // b)
print(a % b)
print(a ** b)
```

```
14
6
40
2.5
2
2
10000
```

Python is PEMDAS compliant

```
print( 2 + 4 * 3 )      ## output is 14
```

Converting a string to an integer

```
s = input("Type an integer:")
i = int(s)              ## i is an integer you can perform calculations on
```

Converting a string to a float

```
s = input("Type a floating point number:")
f = float(s)            ## f is a float you can perform calculations on
```

Converting a number to a string

```
a = 4
s = str(a)
```

OTHER USEFUL NUMBER FUNCTIONS

Random numbers

```
from random import randint          ## top of code
r = randint(0, 9)                   ## where you want
```

3. IF / ELSE

if statement

```
if a > 0:
   print("a is positive")
```

if / else statement

```
if a > 0:
   print("a is positive")
else:
   print("a is negative")
```

If / elif / else

```
if a > 5:
   print("a is 6 or above")
elif a > 0:
   print("a is between 1 and 5")
elif a == 0:
   print("a is zero”)
else:
   print("a is negative”)
```

Nested if

```
if a > 0:
   if b > 0:
      print("a is positive, b is positive")
   else:
      print("a is positive, b is negative")
else:
   if b > 0:
      print("a is negative, b is positive")
   else:
      print("a is negative, b is negative")
```

Multi condition if

```
if a > 0 and b > 0:
   print("a is positive, b is positive")
elif a > 0 and b < 0:
   print("a is positive, b is negative")
elif a < 0 and b > 0:
   print("a is negative, b is positive")
elif a < 0 and b < 0:
   print("a is negative, b is negative")
else:
   print("one of them is zero")
```

Summary of comparison operators

```
if a == b:            # equal
if a != b:            # not equal
if a > b:             # greater than
if a >= b:            # greater than or equal to
if a < b:             # less than
if a <= b:            # less than or equal to
if a == 0 and b == 0: # both are true
if a == 0 or b == 0:  # either is true
```

4. FOR LOOPS

In general: For loops exist to iterative over a list of values, performing a certain task with each individual value in the list.

General rule:

```
for item in sequence:
```

Looping through the characters in a string

```
s = "my string"
for letter in s:
   print( letter )
```

Looping through a range of numbers

```
for num in range(10):
   print( num )        ## will print 0 1 2 3 4 5 6 7 8 9
```

The range() function general rule:

```
range( from, upToButNotIncluding, incrementBy )
```

- The “from” and “incrementBy” are optional. “From” defaults to 0 and “incrementBy” defaults to 1.
- If only one parameter is provided, range() will treat it as the “upToButNotIncluding” number.
- If two parameters are provided, range() will treat them as “from” and “upToButNotIncluding”

Examples of the range() function:

```
range(5)              ## 0,1,2,3,4
range(3,10)           ## 3,4,5,6,7,8,9
range(-4)             ## an empty sequence
range(-4,0)           ## -4,-3,-2,-1
range(0,10,2)         ## 0,2,4,6,8
range(0,-4,-1)        ## 0,-1,-2,-3
```

5. STRINGS

The following examples use this string

```
s = "To infinity and beyond!"
```

Length of a string

```
l = len(s)              ## 23
```

Joining two strings together

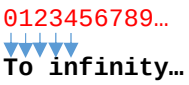
```
newstring = somestring + someotherstring
```

- Use the plus ("+") operator

Getting (slicing) a single character

```
s[4:5]                  # 'n'
```

- From the 4th character (start counting at zero), up to but not including the 5th character.



Getting a substring

```
s[3:11]                 # 'infinity' ... from 3 up to but not including 11
s[:5]                   # 'To in' ... from 0 up to but not including 5
s[5:]                   # 'finity and beyond!' ... from 5 to the end
s[-5:]                  # 'yond!' ... last 5 characters
```

Searching strings

```
s.find("infinity")                                  ## 3
```

- Find the character number (starting from zero) that the substring starts at
- Returns -1 if not found
- If the target appears more than once, find() will provide the starting location of the first occurrence. You can use rfind() to provide the starting location of the last occurrence.

Replacing parts of a string

```
s.replace("infinity", "a hundred")                  ## "To a hundred and beyond!"
s.replace(" ", "@")                                 ## "To@infinity@and@beyond!"
```

- Will replace all occurrences of a match

Counting occurrences within a string

```
s.count("i")                                        ## 3
```

- Will count all occurrences of the target

Restyle of upper / lower casing

```
s.lower()                                           ## to infinity and beyond!
s.upper()                                           ## TO INFINITY AND BEYOND!
s.title()                                           ## To Infinity And Beyond!
s.swapcase()                                        ## tO INFINITY AND BEYOND!
```

Quick queries about the string's contents

```
s.isnumeric()              ## False ... does it contain only numbers?
s.isalpha()                ## False ... does it contain only letters?
s.islower()                ## False ... is it all lowercase?
s.isupper()                ## False ... is it all uppercase?
s.istitle()                ## False ... is it all titlecase?
s.isspace()                ## False ... is it all spaces?
```

Padding with spaces

```
s.ljust(30)                ## "To infinity and beyond!       "
s.rjust(30)                ## "       To infinity and beyond!"
```

- Pad with spaces so it meets the minimum length size

6. WHILE LOOPS

General rule:

```
while condition_is_true:
   keep_looping
```

Whatever conditions work for "if" statements also work for "while" loops.

Example, print the numbers 1 through 10.

```
a = 1
while a <= 10:
   print(a)
   a = a + 1
```

7. LISTS / ARRAYS

In Python, one can store a list of values in one variable and use an index to access individual elements.

```
primes = [ 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
```

To print or change a value in the list:

```
print( days[ 2 ] )                  ## Tuesday
days[2] = "Baumgarday”
print( days[ 2 ] )                  ## Baumgarday
```

To append a new item to the list:

```
primes.append( 31 )
```

For loops work great with lists!

```
for day in days:
   print( day )                     ## Sunday, Monday, Baumgarday, ...
```

If you need to use the index of each item for some reason…

```
for i in range(len(days)):
   print( "Day", i, ”is”, days[i] )
```

A lot of the functionality we know from strings works with lists as well. In fact, Python really just treats strings as a list of characters. For instance:

- How many items in the list?

```
print( len( days ) )                ## 7
```

- Get an individual element of the list

```
bestday = days[ 6 ]
```

- Get a range of elements of the list

```
weekdays = days[ 1:6 ]
weekdays = days[ 1: -1 ]            ## Works just like strings
```

- Does this item exist in the list?

```
print( days.find("Monday") )
```

- Count how many occurrences of the item appear in the list

```
print( days.count("Monday") )
```

- Does the item appear in the list?

```
print( "Tuesday" in days )          ## False
print( "Tuesday" not in days )      ## True
```

Converting between strings and lists of strings

- From a string to a list

```
s = "Hello how are you?"
l = s.split(" ")                    ## Create the list by splitting on spaces
print( l )
```

- From a list to a string

```
days_string = days.join(", ")
print( days_string )
```

Get the maximum or minimum values

```
print( min( primes ) )              ## 1
print( max( primes ) )              ## 29
```

Generators create simple lists automatically for you

- Syntax:

```
newlist = [expression for variable in sequence]
```

- eg: to create a new list of numbers 0, 1, 2, 3, 4

```
nums = [i for i in range(5)]
```

- eg: to create a new list of numbers 0, 10, 20, 30, 40

```
nums = [i*10 for i in range(5)]
```

- is equivalent to the following long-form code

```
nums = []                           ## creates an empty list
for i in range(5):
   nums.append( i*10 )
```

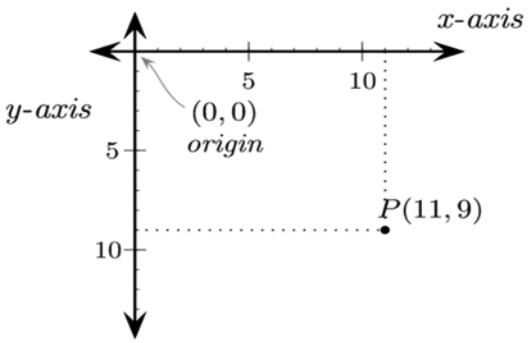
8. PYGAME BASICS

STARTING A TYPICAL PYGAME APP

Creates a PyGame app of dimensions 800 wide x 600 high. "window" is the variable to use for drawing on the PyGame screen.

```
import pygame, sys, random
from pygame.locals import *

pygame.init()
fps = pygame.time.Clock()
window = pygame.display.set_mode((800,600))
```



STRUCTURE & BOILER PLATE

```
import pygame, time, random
from pygame.locals import *

pygame.init()
window = pygame.display.set_mode((500,500))
fps = pygame.time.Clock()

# Declare colors, images, sounds, fonts

# Variables needed for keeping track of positions

# Game will end when this is set to True
quit = False

# Main game loop
while not quit:

    # Process events
    for event in pygame.event.get():
        print(event)
        if event.type == QUIT:
            quit = True
        elif event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                quit = True

    # Perform calculations

    # Draw graphics
    window.fill(BLACK)
    ...
    pygame.display.update() # Actually does the screen update
    fps.tick(25) # Run the game at 25 frames per second

# Loop over, game over
pygame.quit()
```

PYGAME: DRAWING SHAPES

The following functions exist for drawing simple geometric shapes.

If thickness is left out, the shape will be filled in.

- pygame.draw.**line**(window, colour, from-coordinates, to-coordinates, *line-thickness*)
- pygame.draw.**rect**(window, colour, (top-left-x, top-left-y, height, width), *thickness*)
- pygame.draw.**circle**(window, colour, (centre-x, centre-y), radius, *thickness*)
- pygame.draw.**ellipse**(window, colour, (top-left-x, top-left-y, height, width), *thickness*)
- pygame.draw.**polygon**(window, colour, (sets of coordinates for the vertices) , *thickness*)

Example usage:

```
pygame.draw.line(window, BLUE, (50, 60), (50, 160), 2)
pygame.draw.rect(window, GREEN, (52, 160, 120, 40))
pygame.draw.circle(window, WHITE, (110, 110), 40, 1)
pygame.draw.ellipse(window, RED, (220, 100, 80, 40))
pygame.draw.polygon(window, RED, ((20, 20), (52, 60), (171, 60), (200, 20)))
```

PYGAME: DRAWING TEXT

Steps to display text:

1. Create a font variable based on a font name and size
2. Use that font variable to render (draw) some text to an image
   `.render( text, anti-aliasing, colour )`
3. Blit (place) that image of text onto your main window at a set of coordinates

Create a font variable before your game loop. Typically I would add it when all the colours are defined.

```
ARIAL = pygame.font.SysFont("Arial", 60)
```

Inside your loop, before the display.update(), add two lines to render your text to a bitmap, and then blit that bitmap to the screen window.

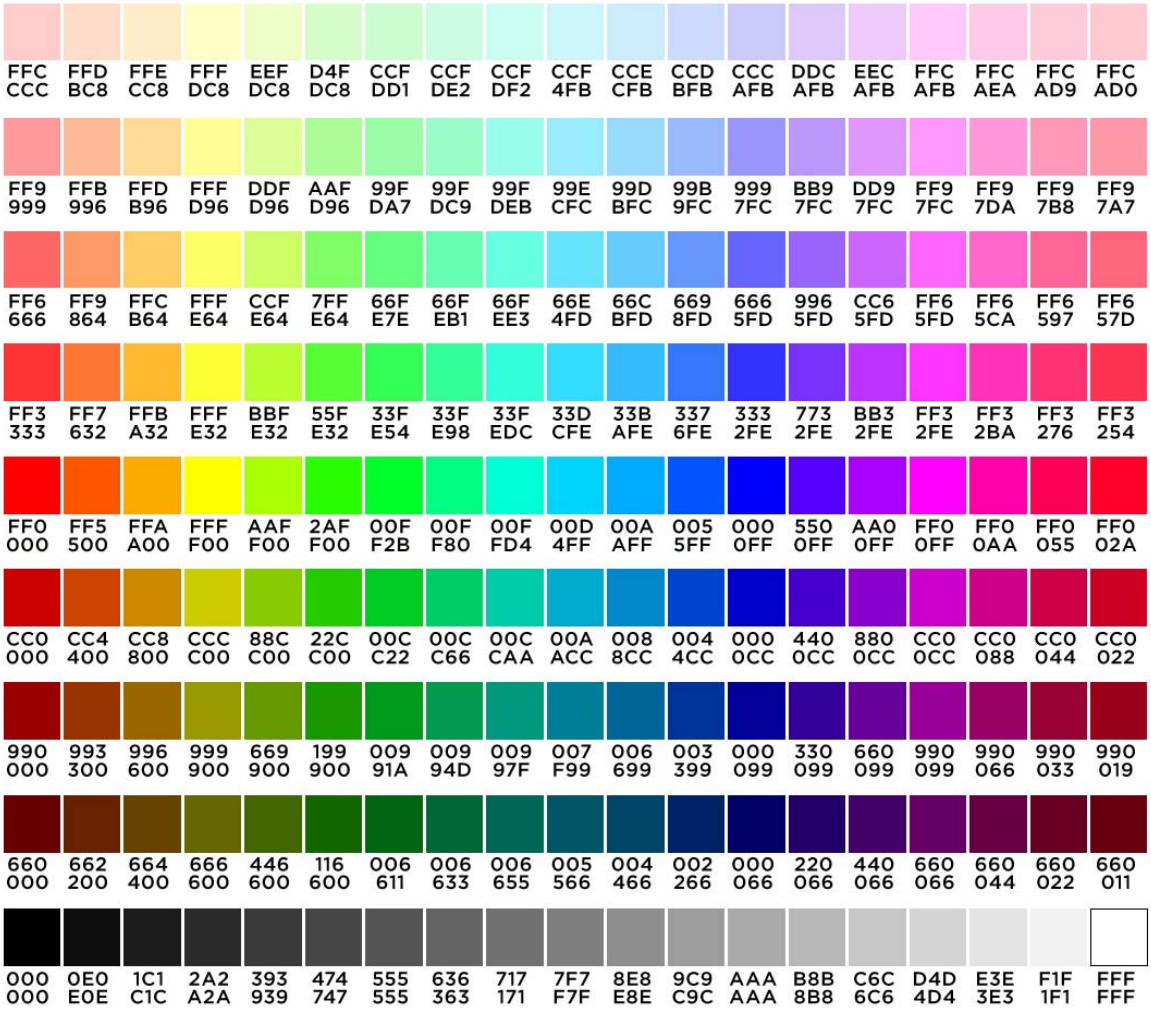
```
label = ARIAL.render("Hello PyGame!", 1, (255,255,0))
window.blit(label, (100, 100))
```

Tip:

- The text you render must be a string. If you have an integer you wish to display (for scores etc), wrap the integer with the str() function first. Eg: **score_string = str(score)**

python™ LANGUAGE REFERENCE

PYGAME: COLOURS



Colours are created using a 3 number variable, COLORNAME = (red, green, blue), where each colour value is between 0 and 255, or 0x00 to 0xFF in hexadecimal. For example:

```
BLACK = (0, 0, 0)              BLACK = (0x00, 0x00, 0x00)
WHITE = (255, 255, 255)        WHITE = (0xFF, 0xFF, 0xFF)
RED   = (227, 27, 27)          RED   = (0xE3, 0x1B, 0x1B)
GREEN = (0, 255, 0)            GREEN = (0x00, 0xFF, 0x00)
BLUE  = (0, 0, 255)            BLUE  = (0x00, 0x00, 0xFF)
```

PYGAME: HELLO PYGAME!

Part 1: Import the libraries, initialize the pygame system, create the window of given dimensions, set a caption

```
import pygame, time
pygame.init()
window = pygame.display.set_mode((600,300))
pygame.display.set_caption("G'day")
```

Part 2: Create some colours

```
YELLOW = (255,255,0)
BLUE = (0,0,255)
GREEN = (0,255,0)
WHITE = (255,255,255)
```

Part 3: Create a font variable, render (draw) some text, blit it onto the screen

```
myfont = pygame.font.SysFont("Arial", 60)
label = myfont.render("Hello PyGame!", 1, YELLOW)
window.blit(label, (100, 100))
```

Part 4: Draw a couple of shapes

```
pygame.draw.line(window, BLUE, (50, 60), (50, 160), 2)
pygame.draw.rect(window, GREEN, (52, 160, 120, 40))
pygame.draw.circle(window, WHITE, (500, 100), 40, 5)
```

Part 5: Update the display, pause for 5 seconds before quitting.

```
pygame.display.update()
time.sleep(5)
pygame.quit()
```

Part 6: Be amazed

PYGAME: EVENTS

CHECKING FOR EVENTS

The most common way to process "events" in PyGame is through a for-loop to run inside your main game loop. This will allow Pygame to process any events that are waiting, or move on if there aren't any.

```
for event in pygame.event.get():
    print(event)
    if event.type == QUIT:
    elif event.type == KEYDOWN:
    elif event.type == KEYUP:
    elif event.type == MOUSEBUTTONDOWN:
    elif event.type == MOUSEMOTION:
```

Each event type also contains information relevant to that event as outlined in each section following…

KEYBOARD EVENTS

If it is a keyboard event…

```
if event.type == KEYDOWN or event.type == KEYUP:
```

Then the "event" object would also contain the following values

- event.key ... contains the numeric code of the key that was pressed/released. The easiest way to check which key it is, is to use the built in K_ variables provided by pygame. Some of the most common key codes follow, otherwise for the full list check https://www.pygame.org/docs/ref/key.html

K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9			
K_a	K_b	K_c	K_d	K_e	K_f	K_g	K_h	K_i	K_j	K_k	K_l	K_m
K_n	K_o	K_p	K_q	K_r	K_s	K_t	K_u	K_v	K_w	K_x	K_y	K_z
K_UP		K_DOWN		K_RIGHT		K_LEFT		K_INSERT		K_DELETE		K_HOME
K_END		K_PAGEUP		K_PAGEDOWN		K_RSHIFT		K_LSHIFT		K_RCTRL		K_LCTRL
K_RALT		K_LALT		K_RETURN		K_ESCAPE		K_SPACE				

MOUSE EVENTS

MOUSEMOTION

- Will trigger any time the mouse moves while it is over your app window.
- Note: Even a 1 pixel movement will trigger the event, so you could get a lot of event signals.
- The "event" object will contain the following useful values:
  - pos will be an (x,y) tuple. Eg:

```
print("The mouse is at x=", event.pos(0), " and y=", event.pos(1))
```

  - buttons is tuple indicating which buttons are currently pressed (left, middle, right). They will be set to True if pressed, False if not.

MOUSEBUTTONDOWN and **MOUSEBUTTONUP**

- Also contains a "pos" tuple with the (x,y) position of the button press/relese.
- Contains a "button" integer with the number of the button pressed (1 = left, 2 = middle, 3 = right)

OTHER EVENTS

The QUIT event is triggered if the user clicks the program close icon.

EXAMPLE: EVENT HANDLING

```
import pygame, time
from pygame.locals import *

pygame.init()
window = pygame.display.set_mode((500,500))
fps = pygame.time.Clock()

# Declare colors
BLACK = (0,0,0)
WHITE = (255,255,255)
YELLOW = (255,255,0)
RED = (255,0,0)
# Variables needed for keeping track of positions
x,y = 250,250
movex = 0
movey = 0
a,b = 0,0
c,d = 0,0
# Game will end when this is set to True
quit = False
# Main game loop
while not quit:
    # Process events
    for event in pygame.event.get():
        print(event)
        if event.type == QUIT:
            quit = True
        elif event.type == KEYDOWN:
            if event.key == K_LEFT:
                movex = -5
            elif event.key == K_RIGHT:
                movex = 5
            elif event.key == K_UP:
                movey = -5
            elif event.key == K_DOWN:
                movey = 5
            elif event.key == K_ESCAPE:
                quit = True
        elif event.type == KEYUP:
            if event.key == K_LEFT or event.key == K_RIGHT:
                movex = 0
            if event.key == K_UP or event.key == K_DOWN:
                movey = 0
        elif event.type == MOUSEMOTION:
            (a, b) = event.pos
        elif event.type == MOUSEBUTTONDOWN:
            (c, d) = event.pos
    # Perform calculations
    x = (x + movex) % 500
    y = (y + movey) % 500
    # Draw graphics
    window.fill(BLACK)
    pygame.draw.circle(window, YELLOW, (a, b), 20, 5)
    pygame.draw.circle(window, WHITE, (x, y), 10)
    pygame.draw.circle(window, RED, (c, d), 10)
    pygame.display.update()

    fps.tick(25)
pygame.quit()
```

EXAMPLE: BASICS OF PONG

```
import pygame, time, random
from pygame.locals import *

pygame.init()
window = pygame.display.set_mode((500,500))
fps = pygame.time.Clock()

# Declare colors
BLACK = (0,0,0)
WHITE = (255,255,255)

# Variables needed for keeping track of positions
ball_x = 250
ball_y = 250
ball_move_x = random.randint(-10,10)
ball_move_y = random.randint(-10,10)
paddle_x = 220
paddle_y = 470
paddle_move_x = 0
paddle_move_y = 0

# Game will end when this is set to True
quit = False

# Main game loop
while not quit:

    # Process events
    for event in pygame.event.get():
        print(event)
        if event.type == QUIT:
            quit = True
        elif event.type == KEYDOWN:
            if event.key == K_LEFT:
                paddle_move_x = -20
            elif event.key == K_RIGHT:
                paddle_move_x = 20
            elif event.key == K_ESCAPE:
                quit = True
            elif event.key == K_SPACE:
                ball_x = 250
                ball_y = 250
                ball_move_x = random.randint(5,10)
                ball_move_y = random.randint(5,10)
        elif event.type == KEYUP:
            if event.key == K_LEFT or event.key == K_RIGHT:
                paddle_move_x = 0

    # Perform calculations
    ball_x = ball_x + ball_move_x
    ball_y = ball_y + ball_move_y
    if ball_x < 0:
        ball_move_x = -ball_move_x
    if ball_x > 500:
        ball_move_x = -ball_move_x
    if ball_y < 0:
        ball_move_y = -ball_move_y
    if ball_y > 500:
        ball_x = 250
```

```
        ball_y = 250
        ball_move_x = 0
        ball_move_y = 0

    paddle_x = paddle_x + paddle_move_x
    if paddle_x < 0:
        paddle_x = 0
    if paddle_x > 440:
        paddle_x = 440

    if Rect(paddle_x, paddle_y, 60, 20).colliderect(Rect(ball_x-5, ball_y-5,10,10)):
        ball_move_y = -abs(ball_move_y)
        ball_move_x = int(ball_move_x * 1.2)
        ball_move_y = int(ball_move_y * 1.2)

    # Draw graphics
    window.fill(BLACK)
    pygame.draw.circle(window, WHITE, (ball_x, ball_y), 10)
    pygame.draw.rect(window, WHITE, (paddle_x, paddle_y, 60, 20))
    pygame.display.update()
    fps.tick(25)

pygame.quit()
```

PYGAME: DETECTING COLLISIONS

Pygame has a built in collision detection function you can use. It works by providing the coordinates to two sets of rectangles, and if there is any overlap it will trigger the collision.

The syntax of an "if" statement checking for a collision might look like:

```
if Rect( x, y, w, h ).colliderect( Rect( x, y, w, h ) ):
   print("There is a collision")
```

The "Basics of Pong" sample code provides an example of it being used.

ADD: Using collision detection with a list of rects as well

PYGAME: IMAGES

The following assumes your image files are located in your PyCharms project folder.

DRAWING AN IMAGE

Load the image to a variable.

Do this only once, typically where you declare your colours, fonts etc. (Every time you run it you are reloading the file into memory, slowing your system down)

```
IMAGE = pygame.image.load("image.jpg").convert_alpha()
```

To then draw the full image onto the screen:

(x,y) is the coordinates to place the top-left corner of the image on your screen.

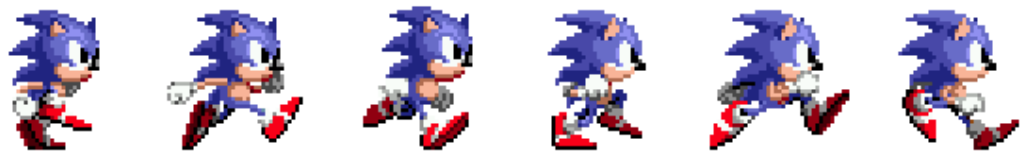
```
window.blit(IMAGE, (x, y))
```

To draw PART of your image onto the screen:

```
window.blit(IMAGE, (window-x, window-y), (image-x, image-y, image-width, image-height))
```

USING A SPRITE MAP

A sprite map is where one image has several icons drawn on it that you may want to use to animate something. For example, to give Sonic the appearance of running, you would cycle through the following images.



Sprite maps are easy to find online. To take the hard work out of using one, put the following code into your project before your colour declarations.

```
def load_sprite_sheet_array(filename, rows, cols, offsetx, offsety):
    images = []
    SS = pygame.image.load(filename).convert_alpha()
    for col in range(0, cols):
        for row in range(0, rows):
            x = row*offsetx
            y = col*offsety
            image = pygame.Surface((offsetx,offsety), pygame.SRCALPHA).convert_alpha()
            image.blit(SS, (0, 0), (x,y,offsetx,offsety))
            images.append( image )
    return images
```

To use your sprite map, you need to open it in Photoshop and measure the number of pixels in width and height for each icon in the map. To load the sprite map as a set of images, add the following code near your colour declarations.

```
SPRITES = load_sprite_sheet_array("sprite-map.png", num-cols, num-rows, width, height)
```

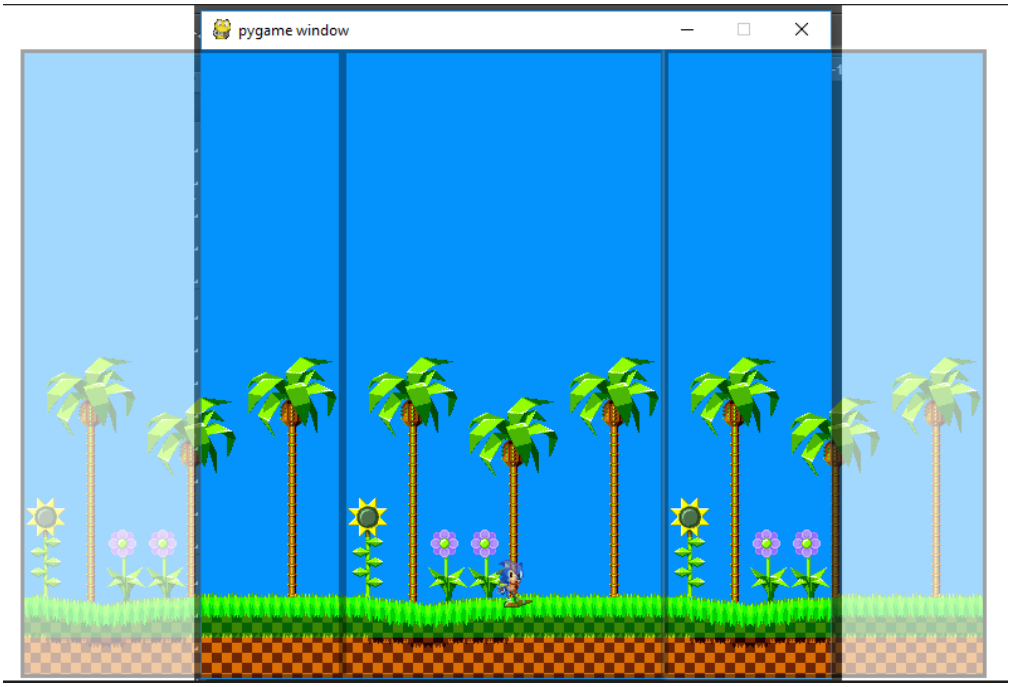
- Num-cols is how many icons horizontally exist in the sprite map
- Num-rows is how many icons vertically exist in the sprite map
- Width is number pixels wide horizontally PER ICON
- Height is number of pixels high vertically PER ICON

To then draw each sprite on screen inside your game loop:

```
window.blit( SPRITES[ spriteNumber ], (x, y) )
```

The spriteNumbers start at 0 and increase horizontally before moving down vertically.

SCROLLING BACKGROUND



The trick to a "scrolling background" is to simply keep redrawing the same image over and over, but at different coordinates each time. As the above illustrates, the scrolling background used in my Sonic spites sample (next page) uses three different images at once, placed side by side.

To give the appearance of scrolling, you need to decide how many pixels per frame your image will move, and then adjust the (x,y) positions by that amount. Once the image has moved so it is no longer on screen, you reset it's location to the starting point of the next background image that needs to emerge on the other side.

In my sample code, tileOffset changes every frame as can be seen here:

```
tileOffset -= 10*runningSpeed
tileOffset = tileOffset % tileWidth
```

The three images are then blited as such:

```
window.blit(TILE, (0 + tileOffset,0))
window.blit(TILE, (tileWidth + tileOffset,0))
window.blit(TILE, (0 - tileWidth + tileOffset, 0))
```

To get this effect to work, it is simply a case of trial and error… adjusting your variables and calculations, try it, adjust something, keep trying until it works the way you want.

SPITES EXAMPLE

Sonic the hedgehog walk, run, sprint with a scrolling background and background music.

```
import pygame, time
from pygame.locals import *

WIDTH = 500
HEIGHT = 500
pygame.init()
window = pygame.display.set_mode((WIDTH,HEIGHT))
fps = pygame.time.Clock()

def load_sprite_sheet_array(filename, rows, cols, offsetx, offsety):
    images = []
    SS = pygame.image.load(filename).convert_alpha()
    for col in range(0, cols):
        for row in range(0, rows):
            x = row*offsetx
            y = col*offsety
            image = pygame.Surface((offsetx,offsety), pygame.SRCALPHA).convert_alpha()
            image.blit(SS, (0, 0), (x,y,offsetx,offsety))
            images.append( image )
    return images

# Declare colours, fonts, sprites and sounds
BACKGROUND = (0x04, 0x92, 0xfc)
SPRITES = load_sprite_sheet_array("soniccd.png", 11, 6, 48, 48)
TILE = pygame.image.load("tile-256x500.png").convert()
tileWidth = 256
tileOffset = 0
pygame.mixer.music.load('03-green-hill-zone.mp3')
pygame.mixer.music.play(-1) # 0 = play once, -1 = loop
JUMPSOUND = pygame.mixer.Sound('Sonic 1/S1_BF.wav')
JUMPSOUND.play()

# Variables needed for keeping track of positions
playerX = WIDTH/2
playerY = HEIGHT-80
runningSpeed = 0
spriteNumber = 0
backgroundSpeed = 0

# Game will end when this is set to True
quit = False

# Main game loop
while not quit:

    # Process events
    for event in pygame.event.get():
        if event.type == QUIT:
            quit = True
        elif event.type == KEYDOWN:
            if event.key == K_RIGHT:
                runningSpeed += 1
            elif event.key == K_LEFT:
                runningSpeed -= 1
            elif event.key == K_ESCAPE:
                quit = True
```

```
    # Perform calculations
    runningSpeed = runningSpeed % 4
    spriteNumber = (spriteNumber + 1) % (len(SPRITES))
    tileOffset -= 10*runningSpeed
    tileOffset = tileOffset % tileWidth

    if runningSpeed == 0: # Standing
        if spriteNumber < 7 or spriteNumber > 10:
            spriteNumber = 7
    elif runningSpeed == 1:  # Walking
        if spriteNumber < 0 or spriteNumber > 6:
            spriteNumber = 0
    elif runningSpeed == 2:  # Running
        if spriteNumber < 11 or spriteNumber > 14:
            spriteNumber = 11
    elif runningSpeed == 3:  # Sprinting
        if spriteNumber < 22 or spriteNumber > 25:
            spriteNumber = 22

    # Draw graphics
    window.fill(BACKGROUND)
    window.blit(TILE, (0+tileOffset,0))
    window.blit(TILE, (tileWidth+tileOffset,0))
    window.blit(TILE, (0-tileWidth+tileOffset, 0))
    window.blit(SPRITES[spriteNumber], (playerX-24,playerY-24))
    pygame.display.update()
    fps.tick(10)

pygame.quit()
```

PYGAME: MUSIC & SOUNDS

Like images, this will assume all your sound files are in your project folder.

BACKGROUND MUSIC

You can only have one track of "background" music playing at a time. You can, however, have multiple sound effects at once.

Playing a background song is dead easy… one command to load it, one command to play.

```
pygame.mixer.music.load('background.mp3')
pygame.mixer.music.play(-1) # 0 = play once, -1 = loop
```

Don't put this in your loop! It should go where colours are declared etc.

SOUND EFFECTS

Make sure you only load the sound effect once. You can use it multiple times, but it will chew up your system memory very quickly if you put the load inside your game loop!

Where you declare your colours etc…

```
BOUNCE_SOUND = pygame.mixer.Sound('sound-effect.wav'))
```

When you want the sound to play

```
BOUNCE_SOUND.play()
```

READ/WRITE FILES (TEXT)

Example reading from a file

- Opens a file for read only purposes
- Removing the end of line characters
- Creates a list of strings, one for each line
- The file "closes" when you end the indentation created by the "with" statement.

```
filename = "days.txt"
days = []
with open(filename, "r") as f:
    for line in f:
        days.append(line.replace("\n",""))
print(days)
```

Example writing to a file

- Opens a file for writing, replacing it if it already existed
- Loops through a list of strings, saving each as a separate line in the file
- Adds the end of line characters to the file
- The file "closes" when you end the indentation created by the "with" statement.

```
with open("days2.txt", "w") as w:
    for day in days:
        w.write(day+"\n")
print("new file created")
```

Based on https://www.digitalocean.com/community/tutorials/how-to-handle-plain-text-files-in-python-3

READ/WRITE FILES (JSON)

```
import json

f = open("countries.json")
data = json.load(f)
print(data)
for datum in data:
    print(datum)
for country in data:
    print("The country code for "+country["name"]+" is "+country["code"])

f.close()
```

JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

https://docs.python.org/3/library/json.html

FUNCTIONS

DICTIONARIES

CLASSES

MATH FUNCTIONS

TIME AND DATE FUNCTIONS

READ FROM A URL (TEXT)

READ FROM A URL (JSON)

https://docs.python.org/3/library/json.html

READ FROM A URL (HTML)

https://docs.python.org/3/library/html.parser.html

READ FROM A FILE (XLSX)

WRITE TO A FILE (PDF)

TAKE A PHOTO WITH THE WEBCAM

RECORD VIDEO WITH THE WEBCAM

RECORD AUDIO WITH THE MICROPHONE

READ/WRITE TO A DATABASE (SQLITE)

python™ LANGUAGE REFERENCE

HOSTING A WEBSERVER (FLASK)

THREADING

SOCKETS