Python summary for Year 8s

This summary is not intended as a teaching guide but a reminder prompt on how to use some the key parts of Python. Use this **after** doing the relevant lessons.

Updated: February 2019 by P.Baumgarten

Print and input

To print text to screen:

```
print("hello")
```

To print a variable to screen:

```
name = "Han Solo"
record = 12
print(f"{name} completed the Kessel run in {record} parsecs")
# notice the 'f' in front of the first set of quotes
```

To ask the user to type input

```
name = input("What is your name? ")                         ## Input saved as text
num = int(input("Type an integer between 1 and 100: "))     ## Input saved as an integer
double = num * 2
print(f"Hello {name}, double your number is {double}")
```

Numbers

Python has two types of numbers, integers and floats. Integers are "whole numbers" without decimals, floats are th name given to numbers that contain decimals.

To get the result of a mathematical calculation, put the equation on the right of an equal sign, and the variable you wish the answer saved in on the left of the equal sign.

Arithmetic

```
a = 100
b = 6
c = a + b           # addition           ... c == 106
c = a - b           # subtraction        ... c == 94
c = a * b           # multiplication     ... c == 400
c = a / b           # division           ... c == 16.66667
c = a // b          # integer division   ... c == 16
c = a % b           # modulus remainder ... c == 4 (ie: remainder of 100 divided by 6)
c = a ** b          # exponent           ... c == 1000000000000 (ie: 10^6)
```

Geometry and trigonometry

```
import math         # add this line to the top of your program for the math functions to work
```

- `math.pi` - returns value of pi with as much precision as available to your computer
- `math.hypot( a, b )` - returns the hypotenuse for a right angled triangle with side lengths a and b
- `math.sin( radians )` - returns the sin() for an angle provided in radians
- `math.cos( radians )` - returns the cos() for an angle provided in radians
- `math.tan( radians )` - returns the tan() for an angle provided in radians
- `math.asin( ratio )` - returns the inverse sin() for a ratio. answer provided in radians
- `math.acos( ratio )` - returns the inverse cos() for a ratio. answer provided in radians
- `math.atan( ratio )` - returns the inverse tan() for a ratio. answer provided in radians
- `math.degrees( radians )` - convert angle from radians to degrees
- `math.radians( degrees )` - convert angle from degrees to radians

Strings

Assign a text string a value

```
s = "Hello"
```

Searching strings

```
s = "To infinity and beyond!"
if "infinity" in s:
   print("Yes, the word infinity is in the string")
else:
   print("No, the word infinity is not in the string")
```

Get substrings

- String positions start from 0. That is, the first letter is position 0, the second letter is position 1 and so forth.
- When asking for a range of characters, Python will give you a substring that includes the starting position number, but not including the end position number.

```
s = "To infinity and beyond!"
s2 = s[:2]                 ## Get from start until position 2. s2 == "To"
s2 = s[16:]                ## Get from position 16 to end. s2 == "beyond!"
s2 = s[3:11]               ## Get from position 3 up to not including position 11. s2 == "infinity"
```

Changing strings

```
s = "To infinity and beyond!"
s2 = s.lower()             ## s2 == "to infinity and beyond!"
s2 = s.upper()             ## s2 == "TO INFINITY AND BEYOND!"
s2 = s.title()             ## s2 == "To Infinity And Beyond!"
s2 = s.swapcase()          ## s2 == "tO INFINITY AND BEYOND!"
s2 = s.ljust(30)           ## s2 == "To infinity and beyond!       "
s2 = s.rjust(30)           ## s2 == "       To infinity and beyond!"
s2 = s.replace(" ", "--")  ## s2 == "To--infinity--and--beyond!"
```

Query content of string

```
s = "To infinity and beyond!"
n = len(s)                 ## get length of string ... n == 23
n = s.count(" ")           ## count spaces in string ... n == 3
n = s.index("o")           ## position of first 'o' in the string ... n == 1
n = s.rindex("o")          ## position of last 'o' in the string ... n == 19
result = s.isnumeric()     ## does it contain only numbers?
result = s.isalpha()       ## does it contain only letters?
result = s.islower()       ## is it all lowercase?
result = s.isupper()       ## is it all uppercase?
result = s.istitle()       ## is it all title case?
result = s.isspace()       ## is it all spaces?
```

If

To execute an "if" statement consists of two parts: Firstly the question you wish to ask, then the code you want to run if the answer to that question is True.

To ask a question, we generally ask Python to compare two or more values to see if they obey a rule.

Examples of number comparisions we can ask include:

```
print( 1 == 1 )            ## Is 1 equal to 1             ... True
print( 1 == 0 )            ## Is 1 equal to 0             ... False
print( "a" == "a" )        ## Is "a" equal to "a"         ... True
print( "a" == "A" )        ## Is "a" equal to "A"         ... False
print( "a" != "z" )        ## Is "a" not equal to "z"     ... True
print( 1 > 0 )             ## Is 1 greater than 0         ... True
print( -1 > 0 )            ## Is -1 greater than 0        ... False
print( 2 >= 3 )            ## Is 2 greater or equal to 3  ... False
print( -3 < -1 )           ## Is -3 less than -1          ... True
print( 3 < 1 )             ## Is 3 less than 1            ... False
print( 2 <= 3 )            ## Is 2 less or equal to 3     ... True
```

Important note: When asking comparisions, we use a double equal sign! A single equal says we want to **set** the value not ask if they are a match.

We can also query string content such as:

```
s = "May the force be with you!"
print(s == "Join the dark side")        ## False... they are not the same
print("the" in s)                       ## True... "the" appears in the content of 's'
```

We can also join multiple queries together such as

```
a = int(input("Enter a number: "))
print( a > 0 and a < 10 )       ## Is number 'a' greater than 0 and less than 10?
print( a < 0 or a > 10 )        ## Is number 'a' less than 0, or is it greater than 10?
```

Once we have our query figured out, we can construct our "if" statement.

```
a = int(input("Enter a number: "))
if (a > 10):
    print("a is bigger than 10")
elif (a > 0):
    print("a is bigger than 0 but not bigger than 10")
elif (a == 0):
    print("a is zero")
else:
    print("a is less than 0")
```

Note, the "if" statement will keep asking questions of the various 'elif' until it finds one that is True. After one item is True, it will skip the rest of the options available and jump to the next thing after the "if" statement is all finished.

While loops

The "while loop" works very similar to the if statement. The difference being that so long as something is True, it will keep running the same indented section of code. An example:

```
up_to = int(input("Enter a number for me to count up to: "))
num = 1
while num <= up_to:
   print( num )
   num = num + 1
print("The end!")
```

For loops

You can also use a for-loop when you know the number of iterations you wish to loop in advance.

```
limit = int(input("Enter a number for me to count up to: "))
for i in range(limit):   # will loop from 0 to up_to-1
   print( i+1 )
print("The end!")
```

You can also specify a starting number other than zero. For instance

```
for i in range(50, 100):   # will loop from 50 to 99
   print( i )
print("The end!")
```

You can even specify that it counts downwards, or using an interval different to one by specifying a third parameter to the `range()` function.

```
for i in range(100, 0, -1):   # will loop from 100 to 1
   print( i )
print("The end!")
```

Lists and for-loops

A list is a means of storing multiple values to one variable name. Other programming languages call these 'arrays'.

Example lists:

```
primes = [1, 2, 3, 5, 7, 11, 13, 17, 19, 23]
vowels = ["A", "E", "I", "O", "U"]
starwars = ["Luke", "Han", "Leah", "Obi-wan", "Yoda", "Rey", "Finn"]
```

Lists have many of the same features of strings (which is really just a list of characters) to query them and get sub-parts from.

```
size = len( starwars )              ## How many items in the starwars list
first = starwars[0]                 ## Get the first item in the starwars list
second = starwars[1]                ## Get the second item in the starwars list
last = starwars[-1]                 ## Get the last item in the starwars list
starwars.append("Darth Vadar")      ## Add an item to the starwars list
"Luke" in starwars                  ## Is "Luke" in the starwars list?
starwars.sort()                     ## Sort the list into alphabetical or numerical order
smallest = min(primes)              ## Get the smallest value from the list
largest = max(primes)               ## Get the largest value from the list
```

We can also use a `for` loop to process every item in a list

```
starwars = ["Luke", "Han", "Leah", "Obi-wan", "Yoda", "Rey", "Finn"]
for character in starwars:
    print(f"{character} is a person in Starwars")
```

For other list functionality such as splitting lists, joining lists, deleting items from a list see https://pbaumgarten.com/python/python-lists/

Turtle

To load the turtle library into Python, add the following to the top of your program file

```
#!/usr/bin/env python3
from turtle import *
```

Moving and Drawing

The turtle will begin in the screen center, facing right. Positive angles rotate counter-clockwise

Command	Example	Description
`home()`	`home()`	return to the starting point and heading
`right( angle-in-degrees )`	`right(45)`	Rotate clockwise the given number of degrees
`left( angle-in-degrees )`	`left(45)`	Rotate counter-clockwise the given number of degrees
`goto( x-coord, y-coord )`	`goto(-50, 50)`	Jump to new x,y coordinates on screen
`setx( x-coord )`	`setx(100)`	Jump only the x coordinate to new position
`sety( x-coord )`	`sety(100)`	Jump only the y coordinate to new position
`setheading( new-angle-in-degrees )`	`setheading(90)`	Point in new direction where 0 == facing right. Positive numbers turn counter-clockwise
`forward( distance )`	`forward(100)`	Move forward given distance of pixels
`backward( distance )`	`backward(100)`	Move backward given distance of pixels
`circle( radius )`	`circle(50)`	Draw a circle with radius 50 pixels
`circle( radius, arc-size-in-degrees )`	`circle(50, 180)`	Draw part of a circle, determined by number of degrees given
`dot( radius )`	`dot(50)`	Draw a filled circle(dot) of given size
`hideturtle()`	`hideturtle()`	Will still draw but hide the little animated turtle shape. Will speed up complex drawings
`showturtle()`	`showturtle()`	Show the turtle when drawing

Pen control

Command	Example	Description
`pendown()`	`pendown()`	Draw including whenever moving, jumping location
`penup()`	`penup()`	Stop drawing when moving
`pensize( width )`	`pensize(1)`	Thickness to draw lines
`isdown()`	`isdown()`	Returns True or False based on if the pen is down

Get turtle information

Command	Example	Description
`position()`	`x,y = position()`	Returns an (x,y) tuple of the current location
`xcor()`	`x = xcor()`	Get the current x-coordinate location
`ycor()`	`y = ycor()`	Get the current y-coordinate location
`heading()`	`direction = heading()`	Get the current facing direction in degrees

Colors

Command	Example	Description
`pencolor( color )`	`pencolor( "yellow")`	Change the pen color
`fillcolor( color )`	`fillcolor( "lime")`	Change the fill color - see begin_fill() and end_fill()!
`begin_fill()`	`begin_fill()`	Tells Python you are starting a shape you want to be filled in when complete
`end_fill()`	`end_fill()`	Tells Python you have finished the shape and to fill it in
`bgcolor( color )`	`bgcolor( "sky blue")`	Change the background color
`bgpic( picture_file )`	`bgpic( "background.gif")`	Set a background picture. Must be GIF format
`bgpic( "nopic" )`	`bgpic( "nopic")`	Removes the background picture

Note: colors can be any of the following:

- A named color, see the list of colour names at https://trinket.io/docs/colors
- A color code in the in form of "#rrggbb", use the google picker at https://www.google.com/search?q=color+picker
- A turple of (red, green, blue) values from 0 to 255 each

Screen settings

Command	Example	Description
`screensize( width, height )`	`screensize( 640,480 )`	Set width and height of turtle screen
`title( name )`	`title( "My amazing project" )`	Set title name of turtle screen
`reset()`	`reset()`	Clear screen, re-center turtle, reset heading to right
`clear()`	`clear()`	Clear screen without recentering turtle or resetting heading
`window_width()`	`w = window_width()`	Get screen width
`window_height()`	`h = window_height()`	Get screen height
`isvisible()`	`vis = isvisible()`	Is the turtle visible?
`speed( new-speed )`	`speed(10)`	Set drawing speed between 1 and 10. Normally starts at 6.
`bye()`	`bye()`	Close turtle
`exitonclick()`	`exitonclick()`	Tells Turtle to quit if the exit icon of the screen is clicked

Events

Command	Example	Description
`onscreenclick( function )`	`onscreenclick( click )`	execute function when screen clicked. callback must take two parameters for x,y coordinates of the click
`onrelease( function )`	`onrelease( click )`	execute function when mouse click let go. callback must take two parameters for x,y coordinates of the click
`onkeypress( function, key )`	`onkeypress( pressed, "Up" )`	execute function when nominated key is pressed. callback must take two parameters for x,y coordinates of the click
`ontimer( function, time-in-ms )`	`ontimer( ticktock, 1000 )`	execute function once after given number milli-seconds
`mainloop()`	`mainloop()`	Start the main event handling loop to run your game

Write text to screen

Command	Example	Description
`write( text )`	`write( "Hello there")`	Write text to screen where ever the turtle is
`write( text, font=(fontname,size,weighting)`	`write( "Hello there", font=("Arial",10,"normal"))`	Write text to screen of specified font

Input popup prompts

Command	Example	Description
`textinput("title", "prompt")`	`name = textinput("Name", "What is your name?"`	Popup box for text information
`numinput("title", "prompt")`	`num = numinput("Enter a number", "Enter a number between 0 and 100")`	Popup box to enter a number

Pygame

Remember that unlike Turtle which has 0,0 in the centre of the screen; Pygame places 0,0 at the top-left of your screen and that an increase in "y" values go down the screen rather than up.

Pygame structure

All Pygames in my tutorials are built from this basic template. Please divide your games into these sections to make it easier to build and for me to assist you with any problems.

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

pygame.init()
window = pygame.display.set_mode((500,500))     # set screen width,height
fps = pygame.time.Clock()

#********** Declare colors, images, sounds, fonts, variables **********
BLACK = (0,0,0)
quit = False
""" insert your code here """

#********** Main game loop starts **********
while not quit:
    window.fill(colors.black)           # Reset the screen to black background
    #********** 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
            """ insert your code here """

    #********** Perform calculations **********
    """ insert your code here """

    #********** Draw graphics **********
    """ insert your code here """

    #********** Update screen **********
    pygame.display.update()         # Actually does the screen update
    fps.tick(25)                    # Run the game at 25 frames per second

#********** Game over **********
pygame.quit()
```

Pygame shapes, colors & text

Note: thickness is optional. If not provided, pygame will fill (color in) the shape. If it is provided, pygame will only draw an outline based on the thickness number.

```
# Rule for a line
pygame.draw.line( window, color, ( x1, y1 ), ( x2, y2 ), thickness )
# Example
pygame.draw.line( window, colors.blue, (50, 60), (50, 160), 10)

# Rule for a rectangle
pygame.draw.rect( window, color, ( x, y, width, height ), thickness )
# Example
pygame.draw.rect( window, colors.green, (52, 160, 120, 40) )

# Rule for a circle
pygame.draw.circle( window, color, ( x, y ), radius, thickness )
# Example
pygame.draw.circle( window, colors.white, (110, 110), 40, 10)

# Rule for an ellipse (oval)
pygame.draw.ellipse( window, colour, ( x, y, width, height ), thickness )
# Example
pygame.draw.ellipse( window, colors.fuchsia, (220, 100, 80, 40) )

# Rule for a multipoint polygone
pygame.draw.polygon( window, colour, ( (x1,y1), (x2,y2), (x3,y3), etc ) , thickness)
# Example
pygame.draw.polygon( window, colors.red, ((20,20), (52,60), (172,60), (200,20)), 5)
```

The built in colors list is:

```
colors.white =     (0xFF, 0xFF, 0xFF)
colors.silver =    (0xC0, 0xC0, 0xC0)
colors.gray =      (0x80, 0x80, 0x80)
colors.black =     (0x00, 0x00, 0x00)
colors.red =       (0xFF, 0x00, 0x00)
colors.maroon =    (0x80, 0x00, 0x00)
colors.yellow =    (0xFF, 0xFF, 0x00)
colors.olive =     (0x80, 0x80, 0x00)
colors.lime =      (0x00, 0xFF, 0x00)
colors.green =     (0x00, 0x80, 0x00)
colors.aqua  =     (0x00, 0xFF, 0xFF)
colors.teal =      (0x00, 0x80, 0x80)
colors.blue =      (0x00, 0x00, 0xFF)
colors.navy =      (0x00, 0x00, 0x80)
colors.fuchsia =   (0xFF, 0x00, 0xFF)
colors.purple =    (0x80, 0x00, 0x80)
```

To get the color for any pixel

```
pixel_colour = window.get_at(( x , y ))
```

Text

Before you can write text, you must create a font variable. Put this in the section of the code template for declaring fonts.

```
ARIAL36 = pygame.font.SysFont("Arial", 36)       ## Font Arial, size 36pt
```

Then in your draw graphics section of the code template, use the following structure to write text.

```
# Rule
window.blit( font_variable.render( text, 1, color_code ), (x, y) )
# Example
window.blit( ARIAL36.render( "Hello Python!", 1, colors.white ), ( 300, 50 ) )
```

Pygame mouse events

```
for event in pygame.event.get():
    if event.type == QUIT:
        quit = True
    elif event.type == MOUSEMOTION:
        x,y = event.pos
        print("You moved your mouse to x={x}, y={y}")
    elif event.type == MOUSEDOWN:
        x,y = event.pos
        print("You clicked your mouse at x={x}, y={y}")
```

For more info, see https://pbaumgarten.com/python/pygame-mouse-events/

Pygame keyboard events

```
for event in pygame.event.get():
    if event.type == QUIT:
        quit = True
    elif event.type == KEYDOWN:
        k = keys.get_key_value(event.key)
        print("You pressed the key {k}")
        if k == "UP":
            print("Going up")
        elif k == "DOWN":
            print("Going down") # .... etc
    elif event.type == KEYUP:
        k = keys.get_key_value(event.key)
        print("You released the key {k}")
```

For normal keys, the `keys.get_key_value()` function will give you the letter or number of the key.

For special keys, the values will be one of the following:

- "LEFT", "RIGHT", "UP", "DOWN", "ENTER", "ALT", "CTRL", "SHIFT", or "DELETE"

For more info, see https://pbaumgarten.com/python/pygame-keyboard-events/

Pygame images & sounds

Drawing an image file (jpeg or png) is really easy! Only two lines of code needed. The following assumes your image files are located in your PyCharms project folder.

Load the image to a variable. Do this only once, typically where you declare your colours, fonts etc. **It is important that your pygame.image.load is not in your game loop. 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 draw the full image onto the screen, use the `blit()` command. The coordinates are the top-left corner of the image on your screen. Do this where you would have used `pygame.draaw.rect` or similar.

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

Resize an image

To resize an image before drawing it onto the screen.

```
picture = pygame.image.load(filename)
picture = pygame.transform.scale(picture, (newWidth, newHeight))
```

Rotate an image

- Will rotate counter-clockwise. Use a negative number to rotate clockwise.
- Unless rotating by 90 degree increments, the image will be padded larger to hold the new size. If the image has pixel alphas, the padded area will be transparent. Otherwise pygame will pick a color that matches the Surface colorkey or the topleft pixel value.

```
originalPicture = pygame.image.load(filename)
rotatedPicture = pygame.transform.rotate(originalPicture, 90)
```

Draw part of an image

To only render part of an image onto the screen, you can supply the coordinates of the rectangle within the image you want to use.

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

Where

- window-x, window-y: the coordinates where you want the partial image located on the screen
- image-x, image-y: within the image file, this is the top left of the part of the image to include
- image-width, image-height: the number of pixels wide and high to include

Animated sprite

Designed especially for sprites created with piskelapp.com.

Prerequisite library

```
pip install pygamemadeeasy
```

Required import statement

```
from pygamemadeeasy import *
```

Create the sprite animation object

```
animation = SpriteAnimation("animation.png", 32, 24) # each frame is 32 x 24
```

Use `object.next_frame()` to return the next frame as an image that is ready to blit. Each time you call the function it will increment to the next frame, and then sequence back to the start of the set.

```
window.blit(animation.next_frame(), (x, y))
```

Background music

Playing a background song is dead easy… one command to load it, one command to play. Don't put this in your loop! It should go where colours are declared etc.

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

It really is important that your pygame.mixer.music.load is not in your game loop. Every time you run it you are reloading the file into memory, slowing your sstem down!

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!

To find good sound effects online, I like freesound.org

Create a sound effect variable…

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

Then in your main game loop, when you want the sound to play…

```
bound_sound.play()
```

Note: Sound effects have to be WAV files.

Pygame collisions

Pygame has a couple of really handy built in collision detection functions you can use: `colliderect` and `collidelist` are the main two I'll discuss here.

Collide with a rectangle

It works by providing the coordinates to two sets of rectangles, and if there is any overlap it will trigger the collision. We've drawn rectangles using `pygame.draw.rect` but you can also create rectangle variables. These can then be used for collision detection as well as drawing on screen (though once you start using images/sprites you probably won't even draw them).

To create a rectangle variable

```
my_rect = Rect( x, y, width, height )
```

To then draw that rectangle use

```
pygame.draw.rect( window, COLOR, my_rect )
```

To check if two rectangles overlap (collide) at all, the "if" statement would look like this:

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

Example code with a couple of rectangle variables would be:

```
ball = Rect( ball_x, ball_y, 20, 20 )
paddle = Rect( paddle_x, 470, 60, 20 )

if ball.colliderect( paddle ):
   print("Collision detected")
```

See the full example at pbaumgarten.com/python/pygame-collisions/

Collide with a list of rectangles

This works very similar as colliderect, the difference being that instead of checking if one rectangle is overlapping another single rectangle, it can check to see if a rectangle is overlapping any items in a list of rectangles!

This is very useful in a game scenario where you might have multiple enemies, bombs or bullets to avoid – you can check your player isn't touching any of them in one line of code!

The key difference is that instead of returning `True` or `False`, the function will return a number to indicate which item in the list there is collision with, or -1 if there is no collision.

A crude illustration of it's use might look like:

```
baddies = [
   Rect( 0, 50, 50, 50 ),
   Rect( 50, 200, 50, 50 ),
   Rect( 200, 100, 50, 50 )
]
me = Rect( 75, 25, 50, 50 )
if me.collidelist(baddies) >= 0:
   print("Uh oh! A baddie has caught you!")
```

See the full example at pbaumgarten.com/python/pygame-collisions/