python™ cheat sheet!

June 2021

Variables

A variable is a name you are assigning to a location of computer memory.

- Variable names can only consist of letters, numbers or underscores. Names are case-sensitive.
- There are four common types of variables: Integers, floats, strings and booleans.

```
n = 42                          # This is an integer ... whole numbers
pi = 3.14                       # This is a float ..... numbers with decimals
event = "Code for Life"         # This is a string ..... "text"
coding_rocks = True             # This is a boolean .... True or False
```

Printing variables

```
print(event)                    # Print just the variable
print("I am attending",event) # Print the text followed by the variable
```

Inputting variables (ask the user to type a value so it doesn't have to be fixed in code)

```
n = int(input("Type an integer:"))
f = float(input("Type a float:"))
t = input("Type some text:")
b = bool(input("Type True or False:"))
```

Calculations

You can perform arithmetic with the numeric variables (integers and floats).

```
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 (remainder of 100 divided by 6)
c = a ** b          # exponent          ... c == 1000000000000 (ie: 10^6)
print(c)
```

You can "add" strings together as well.

```
given_name = "Harry"
surname = "Potter"
full_name = given_name + " " + surname
print(full_name)
```

More resources

I have a number of Python resources online that you are welcome to make use of. If you are working on a project and get stuck, feel free to drop into 333 during a break.

- https://pbaumgarten.com/python
- https://youtube.com/pbaumgarten

Strings

Get substrings (parts of strings)

- String positions start from 0. That is, the first letter is position 0, the second letter is position 1, etc.
- Substring range is inclusive of starting position, exclusive of end position.

```
      0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
s = "To infinity and beyond!"
s2 = s[:2]      # From beginning until character #2.............. s2 contains "To"
s2 = s[16:]     # From character #16 to end...................... s2 contains "beyond!"
s2 = s[3:11]    # From character #3 inclusive to #11 exclusive... s2 contains "infinity"
```

Get properties about the 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? (True/False)
result = s.isalpha()        # does it contain only letters? (True/False)
result = s.islower()        # is it all lowercase? (True/False)
result = s.isupper()        # is it all uppercase? (True/False)
result = s.istitle()        # is it all title case? (True/False)
result = s.isspace()        # is it all spaces? (True/False)
```

Change a string

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

Lists

A list allows you to store multiple values with just one variable name.

```
primes = [1, 2, 3, 5, 7, 11, 13, 17, 19, 23]
vowels = ["A", "E", "I", "O", "U"]
characters = ["Harry", "Hermione", "Ron", "Dumbledore", "Voldemort"]
```

To access individual items within the list. *Note: The first item is item #0.*

```
item = characters[0]       # Will put 'Harry' into item
print(characters[1])       # Will print 'Hermione'
characters[2] = "Weasley"  # Will replace item #2, "Ron", with "Weasley"
```

Special list features for the *if statement* and *for loop*.

```
# Is "Hermione" in the list?
if "Hermione" in characters:
   print("Yes!")
```

```
# Loop over every item in the list
for person in characters:
    print(person)
```

Useful functions that work with lists

```
size = len( characters )  # How many items? 5
last = characters[-1]     # Get the last item
characters.append("Snape")# Add to the end
```

```
characters.sort()       # Sort into order
smallest = min(primes)  # Get smallest value
largest = max(primes)   # Get largest value
```

If statements

If statements compare values.

- If the comparison asked is True, Python will execute the indented lines that follow.
- When asking comparisons, use a double equal sign! A single equal says we want to **set** the value not ask if they are a match.

Comparison operators

```
a == b   # Is a equal b?
a != b   # Is a not equal to b?
a < b    # Is a less than b?
a <= b   # Is a less than or equal to b?
a > b    # Is a greater than b?
a >= b   # Is a greater than or equal to b?
```

Basic "if" statement

```
a = int(input("Type integer a:"))
b = int(input("Type integer b:"))
if a == b:
   print("a and b are the same")
```

More complex "if" statement
('elif' means 'else if', another alternate for a secondary 'if')

```
a = int(input("Type integer a:"))
b = int(input("Type integer b:"))
if a == b:
   print("a and b are the same")
elif a < b:
   print("a is lower than b")
else:
   print("a is higher than b")
```

Loops

There are two main types of loops:

- *While loops* will run continuously while the comparison asked is True.
- The comparison of a *while loop* is designed the same as those for *if statements*.
- *For loops* will run a specific number of times iterating through a sequence of values.

Indentation determines which code will be repeated as part of the loop. End the indentation to return to normal code.

```
until = int(input("Count until?"))
num = 1
while num <= until:
   print( num )
   num = num + 1
print("The end!")
```

```
until = int(input("Count until?"))
# will loop from 0 to (until-1)
for i in range(until):
   print( i )
print("The end!")
```

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

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

Functions

Functions allow you to create reusable blocks of code. They simplify the process of debugging (as you only have to update your code once). The *'def'* keyword defines a function. When Python encounters a function call, it will jump to the code containing the function, execute that, then return to the place it was at in the original code. The return statement indicates the value to send back to the originating code. Python will exit a function when it encounters a return statement (even if you still had more code in the function).

This function requires two values to run. They will be loaded into 'p' and 'q' which will exist for the lifetime of the function.

```
def addition(p, q):
   answer = p + q
   return answer

a,b = 10, 17
c = addition(a, b) # Go to function, sending a & b to p & q, put return value into c.
d = addition(20, 13) # Go to function, sending 20 & 13 to p & q, put return value into d.
print(c, d)
```

Exceptions

```
print("~~~ Division calculator ~~~")
try:
    # Start code that may generate exceptions
    a = float(input("Enter numerator: "))       # <-- possible exception (convert to float)
    b = float(input("Enter denominator: "))     # <-- possible exception (convert to float)
    ans = a / b                                 # <-- possible exception (divide by zero)
    print("The division is: ",ans)
except ValueError:
    print("Can't convert that to a number")
except ZeroDivisionError:
    print("I can't divide by zero, go use Javascript instead ;-)")
except Exception as e: # Other, unexpected error
    # This is not recommended!
    # Syntax errors, name errors will get caught here and you won't realize you have a bug.
    # Catching exceptions should be limited to run-time issues not coding errors.
    print(e) # Print Python's error explanation
```

Dates & times

Creating date/time objects

```
from datetime import datetime

# Create date/time object for 29/7/2021 7:08 pm
# -- Method 1 -- Create from integers
d = datetime(2021, 7, 29, 19, 8, 0)
# -- Method 2 -- Create from parsing a string
d = datetime.strptime("29/07/2021 19:08:00", "%d/%m/%Y %H:%M:%S")

# Create date/time object from computer clock
now1 = datetime.now()               # Local time
now2 = datetime.utcnow()            # UTC time
```

Creating strings to display the date/time objects in a user-friendly manner

```
# Create strings to display the date/times
s1 = d.strftime("%A %d %B, %Y")     # Thursday 29 July, 2021
s2 = d.strftime("%I:%M %p")         # 7:08 pm
print(s1, s2)
```

Codes for datetime strings

%a	Thu	day
%A	Thursday	day
%d	29	day number
%m	07	month
%b	Jul	month
%B	July	month
%y	21	year
%Y	2021	year
%I	7	hour 12h
%H	19	hour 24h
%M	08	minute
%S	00	second
%p	PM	AM/PM

Timestamp conversion

```
# To/from epoch timestamps
# (seconds since 1/1/1970 00:00 UTC)
ts = datetime.timestamp(d)       # 1627556880.0
d = datetime.fromtimestamp(ts)

# Get current epoch timestamp
# -- Method 1 -- using datetime
from datetime import datetime
ts = datetime.now().timestamp()  # 1625564993.5

# -- Method 2 -- using time
import time
ts = time.time()                 # 1625564993.5
```

Time delta – Differences between dates and times

```
from datetime import datetime, timedelta
# Difference between two dates
d = datetime(1969, 7, 20, 20, 17, 40)
difference = datetime.now() - d
print( difference.days )

# What is/was the date X days from now?
now = datetime.now()
past = now - timedelta(days=1000)
future = now + timedelta(days=1000)
```

Random

```
import random

numbers = [0,1,2,3,4,5,6,7,8,9]
weights = [1,1,1,1,1,2,2,2,2,2]
random.seed()                              # initialise function that helps ensure uniqueness
random.shuffle(numbers)                    # Shuffle the list into a random order
pick = random.choice(numbers)              # Select an item from the list
pick = random.randint(0,9)                 # Generate an integer 0 to 9 inclusive
pick = random.choices(numbers, weights) # Use the weights to vary the likelihood of
                                           selecting an item from numbers
```

Dictionaries

Dictionaries are similar to lists except rather than using integer indexes, they use key-value pairs.

```
person = {}                                                # Create empty dictionary
person = {"given": "Paul", "surname": "Baumgarten"}        # Create dict with initial values
person["email"] = "pab@shatincollege.edu.hk"               # Add key-value pair to existing dict
given = person["given"]                                    # Get value from dict
person.pop("email")                                        # Remove email key/value from dict
for key,val in person.items():                             # Iterate over the dictionary
    print(key,"has value",val)
if "email" in person:                                      # Does key exist in the dictionary?
    print(person['given'],"has an email of ",person['email'])
```

```
# Take a dictionary and convert into two lists
keys = person.keys()                                       # Get a list of all keys
vals = person.values()                                     # Get a list of all values
# Take two lists and convert into a dictionary
k = [1,2,3,4,5,6,7]
v = ['sun','mon','tue','wed','thu','fri','sat']
days = dict(zip(k,v))
# days == {1: 'sun', 2: 'mon', 3: 'tue', 4: 'wed', 5: 'thu', 6: 'fri', 7: 'sat'}
```

Log files

Instead of filling the console with the output of a lot of debug print() statements, it is generally better to use a log file for debugging purposes. It is easier to scroll back over previous executions that way.

```
import logging
logging.basicConfig(filename='myproject.log', level=logging.DEBUG)
```

You can change the level of detail written to the log file by changing the level parameter in basicConfig().

Unlike print, the logging functions expect strings, so it is recommended to use f-strings for logging variables.

```
logging.info(f"Program started {datetime.now()}") # Log an info message
logging.debug(f"this is a debug message")         # Log a debug message
logging.debug(f"The value of num: {num}")
logging.error(f"A major error occurred")          # Log an error message
```

The default level is WARNING, which means that only events of this level and above will be tracked, unless the logging package is configured to do otherwise.

You can optionally change the information recorded with every message. For instance to add the time and date to every log entry, add the following parameter:
`basicConfig(format='%(levelname)s %(asctime)s %(message)s')`
https://docs.python.org/3/library/logging.html#logrecord-attributes

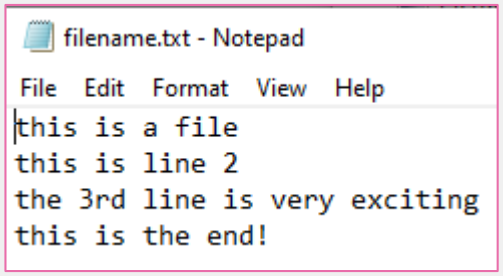
The logger levels are: CRITICAL, ERROR, WARNING, INFO, DEBUG

File read/write basics

It can be very useful to save information to files and read it back later. This will allow your program to "remember" information the next time it is run!

Text files are the most basic types of files. You can create/edit these with Notepad.

```
# Read text file, converting each line to a string,
# resulting in a list of strings.
with open("filename.txt", "r") as f:
    info = f.read().splitlines()

print(info)

# Write a text file. Given `info` is a list of strings,
# save each string to one line of a text file.
with open("filename.txt", "w") as f:
    for line in info:
        f.write(line+"\n")      # The "\n" adds the end of line character
```




```
['this is a file',
'this is line 2',
'the 3rd line is very exciting',
'this is the end!']
```

CSV files are the next most common type of file. CSV = Comma separated values (open it in Notepad to see).
Excel and Google Sheets can easily save as a CSV file for you.

```
import csv

# Read CSV file into a list of dictionaries (a variable containing key/value pairs)
with open("filename.csv", "r", encoding="utf-8") as f:
    records = list(csv.DictReader(f, delimiter=","))

# Print everything
print(records) # see screenshot 2

# Loop through the records
total = 0
for row in records:
    # Do something with the data (need to convert Balance from strings to integers)
    print(row['Name'], "has a balance of:", int(row['Balance'])) # see screenshot 3
    total = total + int(row['Balance'])
print("The total of all balances:", total) # see screenshot 3

# Write CSV file from a list of dictionaries
with open('filename2.csv', 'w', newline='')  as f:
     writer = csv.DictWriter(f, records[0].keys())
     writer.writeheader()
     writer.writerows(records)
```

Screenshot 1:
Excel spreadsheet

	A	B
1	Name	Balance
2	Alice	59
3	Brittany	72
4	Charlotte	83
5	Denise	47

Screenshot 2:
Print of *records*.

```
[ {'Name': 'Alice', 'Balance': '59'},
{'Name': 'Brittany', 'Balance': '72'},
{'Name': 'Charlotte', 'Balance': '83'},
{'Name': 'Denise', 'Balance': '47'} ]
```

Screenshot 3:
Print from for loop.

```
Alice has a balance of: 59
Brittany has a balance of: 72
Charlotte has a balance of: 83
Denise has a balance of: 47
The total of all balances: 261
```

JSON is a commonly used format for exchanging data across the internet. See Requests for more usage.

```
import json

# Read JSON file to list/dictionary structure
with open("data.json", "r") as f:
    data = json.loads(f.read())

# Write list/dictionary structure to JSON file
with open("data.json", "w") as w:
    w.write( json.dumps( data, indent=3 ))
```

Classes

Classes can be thought of as allowing you to define your own datatype. You can group multiple variables together and bind the functions that manipulate them, together into a single unit.

```
import pickle
To use the serialization library
```

```
class Person:
   def __init__(self, givenname, surname):
      self.givenname = givenname
      self.surname = surname

   def get_name(self):
      return self.givenname+" "+self.surname

   def serialise(self):
      with open(self.givenname+"_"+self.surname+".data", "wb") as f:
         pickle.dump(self, f)

   @classmethod
   def deserialise(cls, filename):
      with open(filename, "rb") as f:
         return pickle.load(f)
```

Use the class to create various instance objects and then manipulate them using the attached functions.

```
mrb = Person("Paul", "Baumgarten")
p = Person("Jane", "Doe")
print(p.get_name())
p.serialise() # Save to 'Jane_Doe.data'
jd = Person.deserialise("Jane_Doe.data") # Load object from file
print(jd.get_name())
```

SQLite

```
import sqlite3
def database_write(sql, data=None):
    connection = sqlite3.connect(filename)
    connection.row_factory = sqlite3.Row
    db = connection.cursor()
    if data:
        rc = db.execute(sql, data).rowcount
    else:
        rc = db.execute(sql).rowcount
    connection.commit()
    db.close()
    connection.close()
    return rc # Number of rows affected
```

```
def database_read(sql, data=None):
    connection = sqlite3.connect(filename)
    connection.row_factory = sqlite3.Row
    db = connection.cursor()
    if data:
        db.execute(sql, data)
    else:
        db.execute(sql)
    records = db.fetchall()
    rows = [dict(record) for record in records]
    db.close()
    connection.close()
    return rows # List of dictionaries
```

```
records = database_read("SELECT * FROM people;")
affected = database_write("UPDATE people SET name='test';")
```

```
# Example using dictionary of values as data parameter
inf = {"id":1, "given":"Paul", "familyname":"Baumgarten"}
database_write("UPDATE people SET name=:given, surname=:familyname WHERE id=:id;", inf)
```

Requests

Download JSON data from a website using their API. To know the structure of your request and the resulting data, you will need to review the documentation provided by the individual website service provider.

```
import requests # pip install requests

# Tomorrow's weather forecast for Hong Kong
url = "https://data.weather.gov.hk/weatherAPI/opendata/weather.php"
parameters = {
    "dataType": "fnd",
    "lang": "en"
    }
response = requests.get(url, params=parameters)
if response.status_code == 200: # 200 = Normal/OK
    text = response.text # Get response as a string
    # If response was json, you can convert to list/dictionary structure
    data = response.json()
    day = data['weatherForecast'][0]['week']
    max_temp = data['weatherForecast'][0]['forecastMaxtemp']['value']
    min_temp = data['weatherForecast'][0]['forecastMintemp']['value']
    wind = data['weatherForecast'][0]['forecastWind']
    summary = data['weatherForecast'][0]['forecastWeather']
    print("The weather for",day)
    print("Minimum",min_temp,"Maximum",max_temp)
    print(summary)
    print("Wind:",wind)
```

Uploading files

```
# Simple file upload
f = { 'file': open('report.pdf', 'rb') }
r = requests.post("https://httpbin.org/post", files=f)

# ... or to manually specify a filename and content_type
f = { 'file': ("newname.pdf", open('report.pdf', 'rb'), "application/pdf", {"Expires": "0"} ) }
r = requests.post("https://httpbin.org/post", files=f)

# ... or to upload a string as a file
f = { 'file': ("newname.pdf", data_string, "application/pdf", {"Expires": "0"} ) }
r = requests.post("https://httpbin.org/post", files=f)
```

Downloading binary files

```
import requests, os

# Download a small binary file
response = requests.get("https://cataas.com/cat") # Random cat image
if response.status_code == 200:
    with open('cat.png', 'wb') as f:
        f.write(response.content)
    os.startfile("cat.png") # Windows
    # subprocess.call(('open', filepath)) # Mac # import subprocess

# Download a larger binary file (advised method for >1MB)
url = "https://apod.nasa.gov/apod/image/2105/MWTree_Toledano_6016.jpg"
r = requests.get(url, stream=True) # note the stream=True
chunk_size = 2048 # number of bytes to download at a time
if response.status_code == 200:
    with open('galaxy_tree.jpg', 'wb') as fd:
        for chunk in r.iter_content(chunk_size):
            fd.write(chunk)
    os.startfile("galaxy_tree.jpg") # Windows
```

pillow cheat sheet!

To install package:
pip install pillow

Pillow (Python Imaging Library)

```
# pip install Pillow
from PIL import Image, ImageDraw, ImageFont

# Create new image
img1 = Image.new("RGBA", (1920, 1080))
# Open an image
img1 = Image.open("my picture.png")

# Get image information
width, height = img1.size
mode = img1.mode # Modes: 1 (1bit pixels), L (8bit grey), RGB, RGBA, CMYK, HSV

# Cropping
boundaries = (200, 50, 400, 200)  # x, y, x+w, y+h
img2 = img1.crop(boundaries)

# Resize
img2 = img1.resize((300, 200))

# Rotate (clockwise)
img2 = img1.rotate(45, expand=True, fillcolor="#ff00ff")

# Paste 1 image into another
img1.paste(img2, (100,100)) # Paste at location within img1
img1.paste(img2, (100,100), img2) # Use alpha channel of img2 for transparency of paste
img1.show()

# Change mode
img2 = img1.convert(mode="1") # Convert to black & white

# Drawing on an image
draw = ImageDraw.Draw(img1) # Attach drawing obj to img1
draw.line((x1,y1,x2,y2), fill=(255,255,0), width=1)
draw.rectangle((x,y,width,height), fill="#000000", width=1)
draw.ellipse((x,y,width,height), outline="#ffff00", width=5)

# Get/set individual pixels
img1.getpixel((x,y))
img1.putpixel((x,y), (r,g,b))

# Write text onto image
draw = ImageDraw.Draw(img1)
font = ImageFont.truetype("Roboto-Light.ttf", 48)
draw.text((x,y), "Message", "#ff00ff", font=font)

# Save an image
img1.save("myphoto new copy.png", "png")
# Display image with system viewer
img1.show()
```

Pygame cheat sheet!

June 2021

To install package:
pip install pygame

Basic template/structure

A suggested basic template for a Pygame project

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

pygame.init()                                   # Initialise pygame system
pygame.mixer.init()                             # Initialise audio system
window = pygame.display.set_mode((500,500))  # Window size
fps = pygame.time.Clock()                       # Start the clock

# Declare variables - eg: colors, images, sounds, fonts
black = pygame.Color("#000000")
orange = pygame.Color("#f79052")
pink = pygame.Color("#e96daa")
alive = True
x,y=250,250

# Main game loop
while alive:
    window.fill(black)      # Clear the screen each frame
    # Process events
    for event in pygame.event.get():
        if event.type == QUIT:
            alive = False
        if event.type == MOUSEBUTTONDOWN:
            x,y = event.pos
    # Insert your main game code
    pygame.draw.circle(window, orange, (x, y), 20, 0)
    pygame.draw.circle(window, pink, (x, y), 40, 20)
    # Render the graphics
    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()
```

Detecting key presses

Get a list of all keys pressed

```
keyspressed = pygame.key.get_pressed()
```

If the key is currently being pressed
(ie: will allow for continual behavior while the key is pressed)

```
if keyspressed[ord("a")]:
   print("Pressing A")
if keyspressed[K_RETURN]:
   print("Pressing ENTER")
```

Check for key-press or key-release events.
(ie: will allow for action to occur just once when pressed instead of occurring continually)

```
for event in pygame.event.get():
   if event.type == KEYDOWN:
      if event.key == K_SPACE:
         print("SPACEBAR pressed")
   if event.type == KEYUP:
      if event.key == K_SPACE:
         print("SPACEBAR released")
```

Common key codes for the *event.key* value

```
K_UP         # Up arrow
K_DOWN       # Down arrow
K_LEFT       # Left arrow
K_RIGHT      # Right arrow
K_RETURN     # Return/enter
K_SPACE      # Space bar
K_DELETE     # Delete key
K_a          # Letter 'a'
K_1          # Number '1'
```

Detecting mouse action

```
for event in pygame.event.get():
   if event.type == MOUSEMOTION:      # Mouse movement
      x,y = event.pos
   if event.type == MOUSEBUTTONDOWN: # Mouse click
      x,y = event.pos
      print("clicked at ",x,y)
   if event.type == MOUSEBUTTONUP:    # Mouse release
      x,y = event.pos
      print("released at ",x,y)
```

Playing music and sound effects

To play background music. Use an MP3 file. Should occur before the game loop.

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

To load a short sound effect to a variable. Use a WAV file. Should occur before the game loop.

```
hurt_sound = pygame.mixer.Sound('ouch.wav')
```

To play the sound effect from within your game loop when you want it to occur.

```
hurt_sound.play()
```

Displaying images

Assign a variable the content of an image file. PNG required for transparent backgrounds, otherwise JPG also ok. Perform the load before the game loop.

```
player_image = pygame.image.load("image.jpg")
```

To *"blit"* means to *draw to screen*. Provide x,y coordinates to place top left corner of image.

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

Some useful image modifying functions

```
# Resize to 200 x 100 pixels
resized_image = pygame.transform.scale(original_image, (200, 100))
# Rotate 90 degrees counter-clockwise
rotated_image = pygame.transform.rotate(original_image, 90)
```

To create an animated sequence, load images into a Python list variable

```
# Create a list of images for walking left
player_move = [
   pygame.image.load("walk01.png"),
   pygame.image.load("walk02.png"),
   pygame.image.load("walk03.png"),
   pygame.image.load("walk04.png")
]
# Which frame number we are currently displaying
player_move_frame = 0
```

... then display a different image from the list each time.

```
# Draw the player --- Uses one image from our list of images
window.blit(player_move[ player_move_frame ], (x,y))
# Increment the frame number
player_move_frame = (player_move_frame + 1) % len(player_move)
```

To create your own sprites (image files) for use in a game, I recommend PiskelApp
@ https://www.piskelapp.com/

To download high quality, free for use sprites, I recommend the Kenney collection
@ https://www.kenney.nl/assets/platformer-art-deluxe

Displaying text

Create a variable for the font and size you want to use. Do this before your game loop.

```
arial_24 = pygame.font.SysFont("Arial", 24) # Variable to write Arial size 24pt fonts
```

Use the render() function to create an image based on your message. This can be inside your game loop.

```
label = arial_24.render("Hello Python!", 1, white)  # Create an image containing our text
window.blit(label, (300, 50))                       # Blit the image to screen
```

Drawing shapes

```
# parameters: window, color, (from-coordinates), (to-coordinates), thickness
pygame.draw.line(window, blue, (50, 60), (50, 160), 10)

# parameters: window, color, (x, y, width, height), thickness
pygame.draw.rect(window, green, (52, 160, 120, 40) )

# parameters: window, color, (center x,y), radius, thickness
pygame.draw.circle(window, white, (110, 110), 40, 10)

# parameters: window, color, (x, y, width, height), thickness
pygame.draw.ellipse(window, pink, (220, 100, 80, 40), 10)

# parameters: window, color, (list of coordinate pairs of vertices), thickness
pygame.draw.polygon(window, red, ((20,20), (52,60), (172,60), (200,20)), 5)
```

Detecting collisions

Pygame uses rectangles to detect collisions (checking for any overlap between rectangles).
To detect collision between two rectangles...

```
player = Rect(player_x, paddle_y, 60, 20) # x, y, width, height
enemy  = Rect(enemy_x,  enemy_y,  60, 20) # x, y, width, height
if player.colliderect(enemy):
   print("ouch!")
```

To detect collision between one rectangle and a list of rectangles (such as 1 player and a list of obstacles)

```
player = Rect(player_x, paddle_y, 60, 20) # x, y, width, height
enemies = [
   Rect(100, 100, 60, 20), # x, y, width, height
   Rect(200, 100, 60, 20), # x, y, width, height
   Rect(300, 100, 60, 20), # x, y, width, height
   Rect(400, 100, 60, 20)  # x, y, width, height
   ]
if player.collidelist(enemies) >= 0:
   which_enemy = player.collidelist(enemies)
   print("ouch!")
   print("You hit item",which_enemy,"in the list")
```

To detect collision between one sprite and another sprite, you need to create rectangle variables based on the sprites.
Assuming you have spite code like this before the game loop....

```
player_image = pygame.image.load("goodie.png")
player_location = [0,0] # x,y
enemy_image = pygame.image.load("baddie.png")
enemy_location = [0,500] # x,y
```

... then to detect collision within the loop, it would look like this...

```
# The image.get_rect() function knows the width and height of the image but not it's location
# so that needs to be provided via the parameter setting topleft=(x,y)
player_rect = player_image.get_rect(topleft=player_location)
enemy_rect = enemy_image.get_rect(topleft=enemy_location)
if player_rect.colliderect(enemy_rect):
   print("ouch!")
```

matplotlib cheat sheet!

July 2021

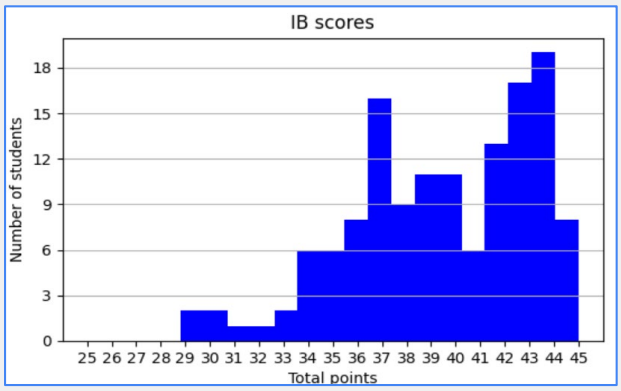
To install package:
```
pip install matplotlib
```

To import into Python:
```
import matplotlib.pyplot as plt
```

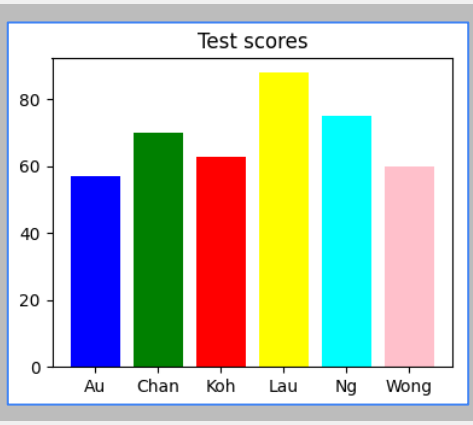
Examples

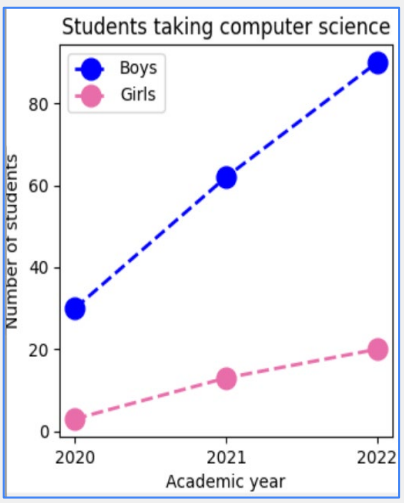
Matplotlib is a powerful chart/graph creating tool in Python.

```
# HISTOGRAM
data = [29,29,30,...] # Extract of data
plt.rcParams["figure.figsize"] = (10,10)
plt.xlabel('Total points')
plt.ylabel('Number of students')
plt.yticks([0,3,6,9,12,15,18,21])
plt.grid(True, axis='y')
plt.title('IB scores')
plt.hist(data, range=(20,45), bins=26, color='blue')
plt.show()
```

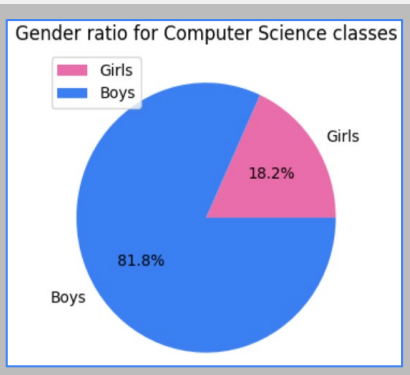



```
# BAR CHART
names = ['Au','Chan','Koh','Lau','Ng','Wong']
scores = [57, 70, 63, 88, 75, 60]
color_list = ['blue','green','red','yellow','cyan','pink']
plt.title('Test scores')
plt.bar(names, scores, color=color_list)
# plt.barh() will give a horizontal bar chart
plt.show()
```



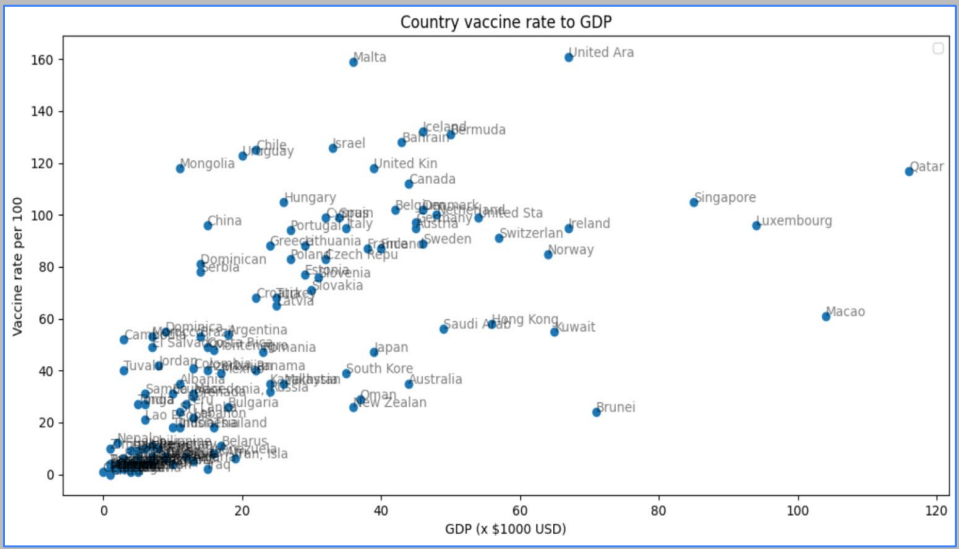


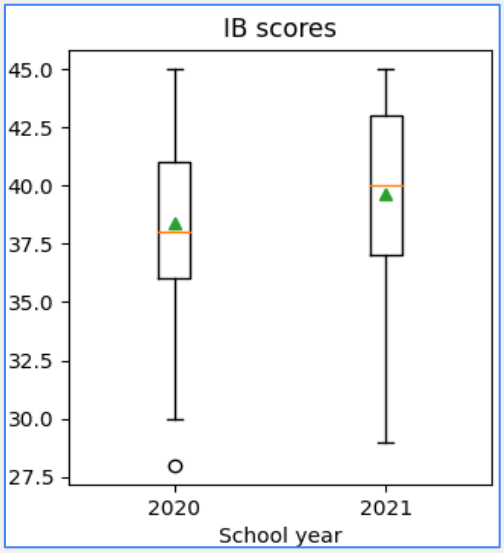

```
# PLOT CHART
years = [2020, 2021, 2022]
boys = [30,62,90]
girls = [3,13,20]
plt.xlabel('Academic year')
plt.xticks(years)
plt.ylabel('Number of students')
plt.title('Students taking computer science')
plt.plot(years, boys, label="Boys", color='blue', marker='o', \
   linestyle='dashed', linewidth=2, markersize=12)
plt.plot(years, girls, label="Girls", color='#e96daa', marker='o', \
   linestyle='dashed', linewidth=2, markersize=12)
plt.legend()
plt.show()
```



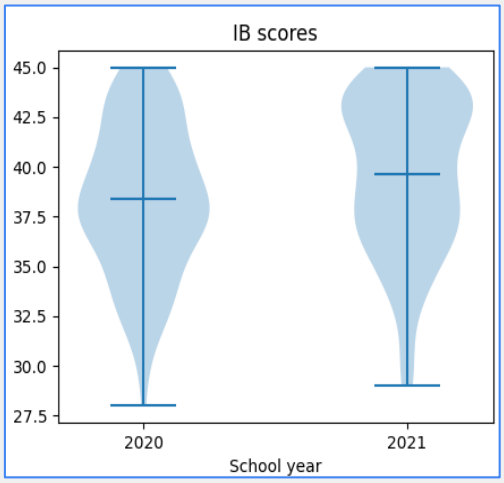

```
# PIE CHART
names = ["Girls","Boys"]
values = [20,90]
colors = ["#e96daa", "#3b7ff2"]
plt.title('Gender ratio for Computer Science classes')
plt.pie(values, labels=names, colors=colors, autopct='%1.1f%%')
plt.legend()
plt.show()
```

Matplotlib examples continued…





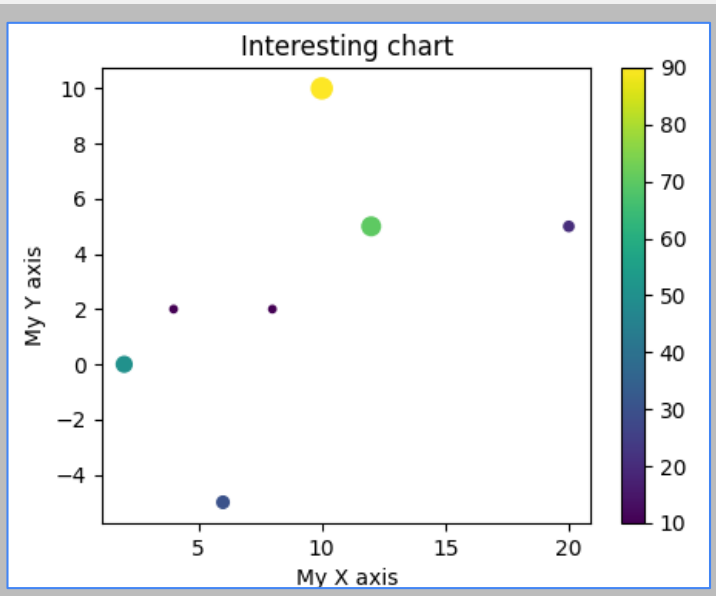

```
# SCATTER DIAGRAM
# Data sourced from ourworldindata.org as of 13/7/2021
labels = ["Australia", .... ] # list of country names
y = [35, ...] # vaccine doses per 100
x = [44, ...] # GDP in 1000s of USD
plt.rcParams["figure.figsize"] = (10,10) # (inches)
plt.xlabel('GDP (x $1000 USD)')
plt.ylabel('Vaccine rate per 100')
plt.title('Country vaccine rate to GDP')
plt.scatter(x,y) # Create chart
for i in range(len(labels)): # label each coordinate
    plt.annotate(labels[i], (x[i], y[i]), alpha=0.5)
plt.show()
```




```
# BOX PLOTS
data2020 = [28,30,30,...] # Extract
data2021 = [29,29,30,...] # Extract
data = [data2020, data2021]
plt.title('IB scores')
plt.xlabel('School year')
ax = plt.gca()
ax.axes.xaxis.set_ticklabels([2020,2021])
plt.boxplot(data, showmeans=True)
plt.show()
```

```
# VIOLIN PLOTS
data2020 = [28,30,30,...] # Extract
data2021 = [29,29,30,...] # Extract
data = [data2020, data2021]
plt.title('IB scores')
plt.xlabel('School year')
ax = plt.gca()
ax.set_xticks([1,2])
ax.set_xticklabels([2020,2021])
plt.violinplot(data, showmeans=True)
plt.show()
```

```
# SCATTER DIAGRAM
# Alternative with size/colour variation
x = [ 2, 4, 6, 8,10,12,20]
y = [ 0, 2,-5, 2,10, 5, 5]
s = [50,10,30,10,90,70,20]
plt.xlabel('My X axis')
plt.ylabel('My Y axis')
plt.title('Interesting chart')
plt.scatter(x, y, s=s, c=s) # s=sizes, c=colors
plt.colorbar()
plt.show()
```



To save plot to image file:
plt.savefig(“file.png“, format: "png“)

cheat sheet!

July 2021

To install package:
```
pip install opencv-contrib-python numpy
```

Examples

Use camera, display video frames, save image to PNG

```
import cv2
import numpy as np

capture = cv2.VideoCapture(0)       # Camera number
capture.set(3, 800)                 # Request camera width
capture.set(4, 600)                 # Request camera height
while True:
    ret, img = capture.read()       # Get image from camera
    cv2.imshow('window label',img)  # Show on screen
    k = cv2.waitKey(100)            # Wait 100 ms for keypress
    if k % 256 == 27:               # If ESC key pressed
        break
cv2.imwrite("final-frame.png", img) # Save final frame as a PNG
```

```
Capture device can also be
a video file...
capture = cv2.VideoCapture(
"video.mp4")
```

Basic face detection algorithm. Requires cascade file from
https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml

```
import cv2
import numpy as np

yellow = (0,255,255)                # Yellow in BGR colors
n = 0                               # Number of faces seen
capture = cv2.VideoCapture(0)       # Camera number
capture.set(3, 800)                 # Request camera width
capture.set(4, 600)                 # Request camera height
cascade_file = "imagestuff\haarcascade_frontalface_default.xml"
cascade = cv2.CascadeClassifier(cascade_file)
while True:
    ret, img = capture.read()       # Get success code and image from camera
    # By default image is BGR (Blue Green Red) rather than RGB - See conversion functions
    # Create gray scale image for face detection algorithm to use
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Detect any faces in the image? Put coordinates of any faces seen in an array
    faces = cascade.detectMultiScale(
        gray,               # Use the grayscale image
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(100, 100)  # Minimum pixel size to recognise as a "face"
    )
    # For every face we found
    for (x,y,w,h) in faces:
        # Draw a rectangle around the face
        cv2.rectangle(img, (x,y), (x+w,y+h), yellow, 2)
        face = img[y:y+h, x:x+w]            # Extract the face portion of the image
        cv2.imwrite(f"face-{n}.png", face)  # Save the face image to disk
        n = n + 1
    cv2.imshow('window label',img)          # Show on screen
    k = cv2.waitKey(100)                    # Wait 100 ms for keypress
    if k % 256 == 27:                       # If ESC key pressed
        break
cv2.imwrite("final-frame.png", img) # Save final frame as a PNG
```

Save to video

```
capture = cv2.VideoCapture(0)           # Camera number
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
video=cv2.VideoWriter('video.mp4',fourcc,25,(800,600))
while True:
    ret, img = capture.read()           # Get success code and image from camera
    frame = cv2.resize(img, (800,600))  # Resize image to 800x600
    video.write(frame)                  # Add frame to video
    k = cv2.waitKey(100)                # Wait 100 ms for keypress
    if k % 256 == 27:                   # If ESC key pressed
        break
video.release()                         # Close video file when done
```

Common tasks

```
# Read an image file
# -- Returns numpy array, containing the pixel values. For colored images, each pixel is represented as
an array containing Red, Green and Blue and optionally Alpha channels.
img = cv2.imread("image.png")
# Read a video file
video = cv2.VideoCapture("video.mp4")
# Read a video stream from camera
video = cv2.VideoCapture(0) # parameter = camera number
# Read an image (frame) from the video file or camera
ret, img = video.read()

# Get image information - For color images
height = img.shape[0]
width = img.shape[1]
channels = img.shape[2] # Will not exist for grayscale

# To get an individual pixel
pixel = img[y, x]

# Resize an image
img2 = cv2.resize(img, (800,600))

# Text on image
font = cv2.FONT_HERSHEY_SIMPLEX
# parameters: image, string message, top/left coordinates, font scale/size, color, font width
cv2.putText(img, "Message", (x,y), 1, yellow, 2)

# Display on screen
cv2.imshow('window label',img)

# Image conversions - examples
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)    # From BGR to grayscale
rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)      # From BGR to RGB
bgr = cv2.cvtColor(img, cv2.COLOR_RGBA2BGR)      # From RGBA to BGR

# Convert PIL image object to CV2 image array
cv2_image = cv2.cvtColor(np.array(pil_image), cv2.COLOR_RGB2BGR)

# Convert CV2 image array to PIL image object
pil_image = Image.fromarray(cv2.cvtColor(cv2_image, cv2.COLOR_BGR2RGB))

# Paste one img on top of another
h,w,channels = img2.shape   # Get dimensions of img2 to paste
x, y=50,50                  # Coordinates to apply paste in img1
img1[y:y+h, x:x+w] = img2   # Will paste without regard to alpha transparency
cv2.imshow("Final product",img1)
```

Flask cheat sheet!

July 2021

To install package:
pip install flask

Basic template/structure

```
from flask import Flask, render_template, request, redirect, send_file

app = Flask(__name__)
app.config['SECRET_KEY'] = 'code used to secure cookies from tampering'

# Return templates/index.html
@app.route("/")
def index_page():
    return render_template("index.html")

# Return a binary file
@app.route('/promovideo')
def promovideo():
    return send_file("promovideo.mp4")

# Use the URL path to supply a parameter
@app.route('/user/<userid>')
def users(userid):
    return "User page for "+userid

# Get values from a HTML form
@app.route("/page2", methods=['GET','POST'])
def page2():
    # HTML with <input name='person'> will create a request.values['person']
    # .values contains .args and .form combined
    form = dict(request.values) # Convert all values into a dictionary
    person = form['person']
    return f"Hello, {person}, welcome to my website"

# Start the web server. These should be the last lines.
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=80, debug=True)
```

File uploads from browser

Ensure the HTML form containing the file upload uses the correct enctype. Example:
`<form action="/upload" method="post" enctype="multipart/form-data">`

```
# from werkzeug.utils import secure_filename
# import os
# app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024

@app.route("/upload", methods=["POST"])
def upload_photo():
    # The name attribute in <input type='file' name='picture'> will be the key in .files[]
    # secure_filename() protects from security risks with a user-supplied filename
    f = request.files['picture']
    f.save(os.path.join(app.root_path, 'photos', secure_filename(f.filename)))
    return 'file uploaded successfully'
```

Receive/send JSON data

```python
@app.route("/api", methods=['GET','POST'])
def api():
    if request.is_json:
        data = request.json # list/dictionary of values
        print(data)
        result = {"status": "received"}
    else:
        result = {"status": "error"}
    return jsonify(result)
```

Jinja2 templating language

Embed jinja2 templating language into your HTML files. Supply variable data to the **`render_template()`** function for parsing.

```python
return render_template("main.html", personName=p, people=names, variable=v)
```


```html
<!DOCTYPE html>
<html>
    <body>
        <!-- Insert content of variable -->
        <h1>Welcome, {{ personName }}!</h1>

        <!-- if statement -->
        {% if personName != "" %}
            <!-- Insert content of variable -->
            <h1>Welcome, {{ personName }}!</h1>
        {% else %}
            <h1>Welcome visitor!</h1>
        {% endif %}

        <!-- for loop -->
        <ul>
        {% for name in people %}
            <li>{{ name }}</li>
        {% endfor %}
        </ul>

        <!-- if variable is defined -->
        {% if variable is defined %}
            <p>variable is defined with value {{ variable }}</p>
        {% endif %}

        <!-- if variable is empty -->
        {% if variable|length %}
            <p>variable is not empty</p>
        {% else %}
            <p>variable is empty</p>
        {% endif %}

        <!-- drop down list: default to the option based on value of `choice` -->
        <select name="day">
            <option {{ "selected" if choice=='Monday' else "" }}>Monday</option>
            <option {{ "selected" if choice=='Tuesday' else "" }}>Tuesday</option>
        </select>

    </body>
</html>
```

Flask-login

A very common use-case is to require logins to access certain pages, or to customize results for the logged in user.
`pip install flask-login`

```
from flask import Flask, render_template, request, redirect
import flask_login, hashlib

app = Flask(__name__)
app.config['SECRET_KEY'] = "shhhh... it's a secret"
login_manager = flask_login.LoginManager()
login_manager.init_app(app)

class User(flask_login.UserMixin):                # Object used to model user information
    def __init__(self, userid, email, name):
        self.id = userid
        self.email = email
        self.name = name
    def get_dict(self):
        return {'userid': self.id, 'email': self.email, 'name': self.name}

@login_manager.user_loader                        # Required function: return a user object
def load_user(userid):
    users = database_read(f"SELECT * FROM accounts WHERE userid='{userid}';")
    user = User(users[0]['userid'], users[0]['email'], users[0]['name'])
    return user

@app.route("/login", methods=['GET'])
def login_page():
    return render_template("login.html", alert="")

@app.route("/login", methods=['POST'])
def login_request():
    form = dict(request.values)
    users = database_read("SELECT * FROM accounts WHERE userid=:userid", form)
    salt = users[0]['salt']
    saved_key = users[0]['password']
    generated_key = hashlib.pbkdf2_hmac('sha256',
       form['password'].encode('utf-8'), salt.encode('utf-8'), 10000).hex()
    if saved_key == generated_key: # passwords match
        user = load_user(form['userid'])
        flask_login.login_user(user)              # Login user
        return redirect("/main")
    else: # Password incorrect
        return render_template("login.html", alert="Invalid user/password. Please try again.")

@app.route("/main")
@flask_login.login_required
def main_page():
    user = flask_login.current_user               # Get object for current user
    return render_template("main.html", user=user.get_dict())

@app.route("/logout")
@flask_login.login_required
def logout_page():
    flask_login.logout_user()                     # Logout user
    return redirect("/")

@app.route("/")
def index_page():
    if flask_login.current_user.is_authenticated:   # Are we logged in?
        return redirect("/main")
    else:
        return redirect("/login")
```

Flask-socketio

```
# pip install flask-socketio
from flask_socketio import SocketIO, emit, join_room, leave_room

# after app = ...
socketio = SocketIO(app, cors_allowed_origins="*", logger=True, engineio_logger=True)

# Executes when a new client connects to the socketio server
@socketio.on('connect')
def connect():
    print('[socketio connect]')

# Executes when a new client disconnects from the socketio server
@socketio.on('disconnect')
def disconnect():
    print('[socketio connect]')

# Executes for any incoming socketio message (useful for testing purposes).
# TODO: Delete once no longer needed for debugging purposes
@socketio.on('message')
def message(data): # data can be a string or json
    logging.info('[socketio message]', data)

# Executes when a socketio message of type `example1` is received
@socketio.on('example1')
def example1_handler(data): # data can be a string or json
    logging.info(f'example1 received: {data}')
    response = {"something": "some other thing"}
    emit("reply", response) # emit a reply named 'reply' with body in `response`

# Executes when a socketio message of type `example1` is received
@socketio.on('broadcast')
def shout(data): # data can be a string or json
    logging.info(f'example1 received: {data}')
    response = {"something": "some other thing"}
    # Send to all clients connected (including the original sender)
    emit("message", response, broadcast=True)

@socketio.on('room')
def room(data): # data can be a string or json
    if data['action'] == "join":
       # Assumes packet {'action': 'join', 'roomid': 'xxx'}
       join_room(data['roomid'])
       emit("message", "You have joined "+data['roomid'])
    if data['action'] == "leave":
       # Assumes packet {'action': 'leave', 'roomid': 'xxx'}
       leave_room(data['roomid'])
       emit("message", "You have left "+data['roomid'])
    if data['action'] == "message":
       # Assumes packet {'action': 'message', 'roomid': 'xxx', 'message': 'xxx'}
       roomid = data['roomid']
       response = data['message']
       emit("message", response, to=roomid)

# Change the webserver initialisation as follows
if __name__ == '__main__':
    # app.run(host="0.0.0.0", port=80, debug=True) # <-- without socketio
    socketio.run(app, host="0.0.0.0", port=80, debug=True)
```

cheat sheet!

July 2021

Basic template/structure

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <!-- Improves mobile phone compatibility -->
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!-- Load any CSS files here -->
        <link rel="stylesheet" type="text/css" href="/static/main.css" media="screen">
        <title>My website title</title>
    </head>
    <body>
        <!-- Your HTML content goes here -->
        <h1>Hello world!</h1>
        <!-- Load any JavaScript files here -->
        <script type="text/javascript" src="/static/main.js"></script>
    </body>
</html>
```

Common elements

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<hr> <!-- horizontal rule -->
<p>Paragraph</p>
<div class="something">Divider (mostly used with css)</div>
<span class="something">Span (mostly used with css)</span>

<!-- Links (anchors) -->
<a href="page2.html">Go to page 2</a>
<a href="page2.html" target="_blank">Go to page 2, opening a new tab</a>

<!-- Unordered list (dot points)-->
<ul>
    <li>Item 1</li> <!-- list item -->
    <li>Item 2</li>
    <li>Item 3</li>
</ul>

<!-- Multi media -->
<img src="picture.png" width="500" height="600" alt="Picture of person">
<audio src="song.mp3" controls autoplay loop muted></audio>
<video src="movie.mp4" width="640" height="480" controls autoplay loop muted></video>

<!-- HTML tables -->
<table>
    <thead><tr>
            <th>Column header 1</th>
            <th>Column header 2</th>
    </tr></thead>
    <tr>
        <td>Value 1</td>
        <td>Value 2</td>
    </tr>
</table>
```

HTML form basic structure

multipart/form-data is required for file uploads

```
<!DOCTYPE html>
<html>
    <body>
        <!-- A basic form -->
        <form action="http://target/webpage" method="POST" enctype="multipart/form-data">
            <p>What is your name?</p>
            <label for="person">Your name</label>
            <input type="text" name="person">
            <label for="submit">Submit</label>
            <input type="submit" name="submit" value="Answer">
        </form>
    </body>
</html>
```

HTML form inputs

```
<!-- Normal text box for input (size=length in characters)-->
<input type="text" name=? size=? placeholder=? value=?>
```

placeholder -> text to display while empty
value -> pre-assign the text value to display

```
<!-- Multi-line textbox -->
<textarea name=? cols="x" rows="y" placeholder=?>value</textarea>
```

```
<!-- Textbox with auto-complete -->
<input type="text" name=? list="day">
<datalist id="day">
    <option value="Monday"/>
    <option value="Tuesday" />
    <option value="Wednesday" />
</datalist>
```

```
<!-- Drop down menu -->
<select name=?>
    <option value="1" selected>January</option>
    <option value="2">February</option>
    <option value="3">March</option>
</select>
```

```
<!-- Buttons -->
<input type="submit" value="Submit">            <!-- Submit button -->
<input type="image" name=? src=? border=? alt=?><!-- Submit button using an image -->
<input type="reset">                            <!-- Reset button to clear all inputs -->
<input type="button" name=? value=?>            <!-- Functionless button (use js) -->
```

```
<!-- Checkbox -->
<input type="checkbox" name=? value=?>
<input type="checkbox" name=? value=? checked>
```

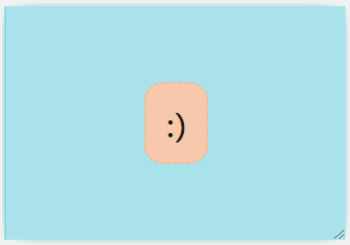
```
<!-- Radio button (multi choice style, only 1 can be picked) -->
<input type="radio" name=? value=?>
<input type="radio" name=? value=? checked>
```

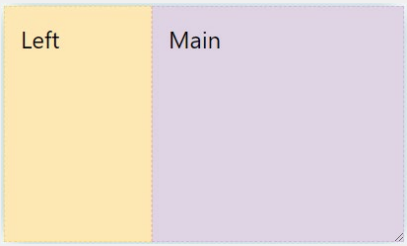
```
<!-- Specialty input types -->
<input type="password" name=? size=?>           <!-- Password entry box -->
<input type="email" name=?>                     <!-- Email address -->
<input type="url" name=?>                       <!-- URL address -->
<input type="tel" name=?>                       <!-- Telephone number -->
<input type="number" name=?>                    <!-- Number picker -->
<input type="range" min="0" max="100" name=?>   <!-- Number picker from 0 and 100 -->
<input type="date" name=?>                      <!-- Calendar date selector -->
<input type="time" name=?>                      <!-- Time selector -->
<input type="color" name=?>                     <!-- Colour picker -->
<input type="file" name=?>                      <!-- Upload a file (see required enctype) -->
```

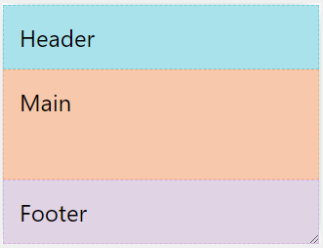
CSS cheat sheet!

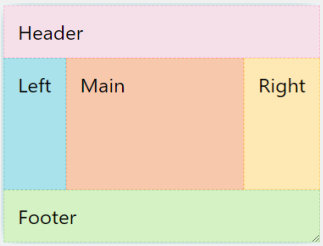
July 2021

Example CSS layouts










```html
<div class="parent" >
    <div class="box">
    :)
    </div>
</div>
```

```css
.parent {
    display: grid;
    place-items: center;
}
```

Layouts from https://1linelayouts.glitch.me/

```html
<div class="parent">
    <div class="section">Left</div>
    <div class="section">Main</div>
</div>
```

```css
.parent {
    display: grid;
    grid-template-columns: minmax(150px, 25%) 1fr;
}
```

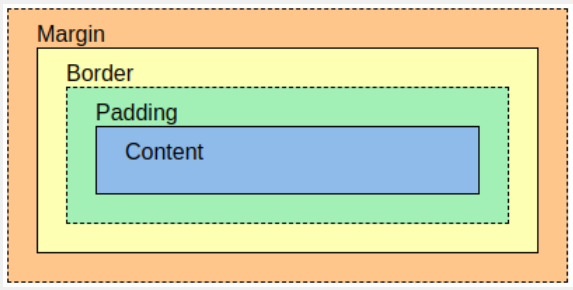
```html
<div class="parent">
    <div class="section">Header</div>
    <div class="section">Main</div>
    <div class="section">Footer</div>
</div>
```

```css
.parent {
    display: grid;
    grid-template-rows: auto 1fr auto;
}
```

```html
<div class="parent">
    <div class="header">Header</div>
    <div class="left-side">Left</div>
    <div class="main">Main</div>
    <div class="right-side">Right</div>
    <div class="footer">Footer</div>
</div>
```

```css
.parent {
    display: grid;
    /* Define the rows / Define the columns */
    grid-template: auto 1fr auto / auto 1fr auto;
}

.header {
    padding: 2rem;
    /* From column 1 up to but not including 4 */
    grid-column: 1 / 4;
}

.left-side {
    grid-column: 1 / 2;
}

.main {
    grid-column: 2 / 3;
}

.right-side {
    grid-column: 3 / 4;
}

.footer {
    grid-column: 1 / 4;
}
```

CSS Box model



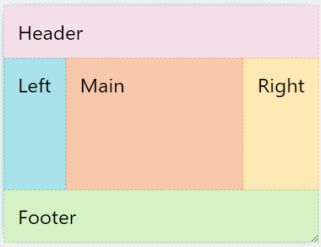
CSS units

px - Pixels
% - Percentage of the enclosing element
vh - 1% increments of the height of the viewport
vw - 1% increments of the width of the viewport
em - Height of the current font spacing.
pt - Points (fonts)
fr - Fractional proportions

CSS common instructions

Example layout that auto switches between Desktop and Mobile screens

```
/* Desktop screen layout */
.container {
    display: grid;
    grid-template-columns:
       minmax(150px, 25%) 1fr;
    gap: 5px 5px;
    height: 100%;
    grid-template-areas:
      "folders main"
}

/* Phone screen layout */
@media screen and (max-width: 800px)
and (orientation: portrait){
    .container {
        grid-template-columns: 1fr;
        gap: 5px 5px;
        height: 100%;
        grid-template-areas:
        "folders"
        "main"
    }
}

.folders {
    grid-area: folders;
}

.main {
    grid-area: main;
}
```



An alternative grid approach to creating this layout through the use of named grid-areas.

CSS selector syntax

```
/* Apply this rule to all <p> tags */
p {
    color: #ff0000;
}

/* Apply this rule to all with class="red" */
.red {
    color: #ff0000;
}

/* Apply this rule where id="special" */
#special {
    color: #ff0000;
}

/* Apply this rule to <input type="password"> */
input[type='password'] {
    color: #ff0000;
}
```

```
html, body {
    height: 100%;
    margin: 0;
}
.parent {
    height: 100%;
    width: 100%;
    display: grid;
    grid-gap: 0;
    grid-template-columns: 200px auto 200px;
    grid-template-rows: 64px auto 64px;
    grid-template-areas:
        "header     header  header"
        "left-side  main    right-side"
        "footer     footer  footer";
}
.header {
    grid-area: header;
    background-color: pink;
}
.left-side {
    grid-area: left-side;
    background-color: cyan;;
}
.main {
    grid-area: main;
    background-color: lightsalmon;
}
.right-side {
    grid-area: right-side;
    background-color: palegoldenrod;
}
.footer {
    grid-area: footer;
    background-color: lightgreen;
}
```

JS cheat sheet!

July 2021

Basic variables & control structures

```
// Variable declarations

let inte = 1;
let floa = 3.14;
let stri = "Hello world!";
let bool = true;

let arra = ["this", "is", "an",
"array"];

let obje = {
    "key": "value",
    "num": 10,
    “doubleNum": function() {
       return (this.num*2)
    }
}
```

```
// Number functions

let pi = 3.141;
// returns "3.14" 2 decimal places
let s = pi.toFixed(2);
// returns "3.1" 2 sig figures
let s = pi.toPrecision(2)
// converts String to number
let i = Number("42");
// milliseconds since 1970
let i = Number(new Date())
// returns the first number: 3
let i = parseInt("3 months");
// returns 3.5
let f = parseFloat("3.5 days");
```

```
// Control structures
let a = 5;
let b = 10;
if (a === b) {
    console.log("a and b are the same");
} else if (a < b) {
    console.log("a is less than b");
} else {
    console.log("b is less than a");
}
// ----------------------------------
let i=0;
while (i<10) {
    console.log(i);
    i++; // increment i by 1
}
// ----------------------------------
let i=0;
do {
    console.log(i);
    i++;
} while (i<10);
// ----------------------------------
for (let i=0; i<10; i++) {
    console.log(i); // iterate i 0 through 9
}
// ----------------------------------
for (let i in arra) { // will iterate over the indexes
    console.log(arra[i]);
}
// ----------------------------------
for (let element of arra) { // will iterate over values
    console.log(element);
}
```

```
// Math
var pi = Math.PI;        // 3.141592653589793
Math.round(4.4);         // = 4 - rounded
Math.pow(2,8);           // = 256 - 2 to the power of 8
Math.sqrt(49);           // = 7 - square root
Math.abs(-3.14);         // = 3.14 - absolute, +ve value
Math.ceil(3.14);         // = 4 - rounded up
Math.floor(3.99);        // = 3 - rounded down
Math.sin(0);             // = 0 - sine
Math.cos(Math.PI);       // OTHERS: tan,atan,asin,acos,
Math.min(0, 3, -2, 2);   // = -2 - the lowest value
Math.max(0, 3, -2, 2);   // = 3 - the highest value
Math.log(1);             // = 0 natural logarithm
Math.exp(1);             // = 2.7182pow(E,x)
Math.random();           // random number between 0 and 1
```

Strings

```
// Strings
let abc = "abcdefghijklmnopqrstuvwxyz";
let esc = 'I don\'t \n know';   // \n new line
let len = abc.length;           // string length
abc.indexOf("lmno");            // find substring, -1 if doesn't contain
abc.lastIndexOf("lmno");        // last occurance
abc.slice(3, 6);                // cuts out "def"
abc.replace("abc","123");       // find and replace
abc.toUpperCase();              // convert to upper case
abc.toLowerCase();              // convert to lower case
abc.concat(" ", str2);          // abc + " " + str2
abc.charAt(2);                  // character at index: "c"
abc[2];                         // unsafe, abc[2] == "C" doesn't work
abc.charCodeAt(2);              // character code at index: "c" -> 99
abc.split(",");                 // split into an array
abc.split("");                  // split on each character
```

Functions

There are two common ways of defining functions.

```
// Method 1 - The "traditional" approach
function adder(a, b) {
    return a + b;
}
```

```
// Method 2 - Arrow functions
const adder = (a, b) => {
    return a + b;
}
```

Arrow functions have become very popular within Javascript and you will see their use in many modern tutorials. Arrow functions do not need to be named. The function body can be included in it's entirety as a parameter when another function expects a function as a parameter. For example, the following two code snippets are equivalent.

```
// Define the function
const processResult = (result) => {
    console.log("The website returned", result.text);
}
// Supply the function as a parameter like a common variable
fetch("https://example.com").then( processResult );
```

```
// Supply function body as a parameter where a function is expected
fetch("https://example.com").then( (result) => {
    console.log("The website returned", result.text);
});
```

Classes

```
class Shape {
    constructor (id, x, y) {
        this.id = id;
        this.move(x, y);
    }
    move (x, y) {
        this.x = x;
        this.y = y;
    }
    pos () {
        return {"x":this.x, "y":this.y};
    }
}
```

```
// Inheritance
class Rectangle extends Shape {
    constructor (id, x, y, w, h) {
        super(id, x, y);
        this.width  = w;
        this.height = h;
    }
    area () {
        return this.width * this.height;
    }
}

// Instantiation
let rect = new Rectangle("rect", 0, 0, 200, 500);
console.log("The area is ",rect.area());
console.log("The coordinates are ",rect.pos());
```

Arrays

```
let dogs = ["Bulldog", "Beagle", "Labrador"];
dogs.length;                            // number of items: 3
dogs.toString();                        // convert to string: "Bulldog,Beagle,Labrador"
dogs.join(" * ");                       // join: "Bulldog * Beagle * Labrador"
dogs.pop();                             // remove last element
dogs.push("Chihuahua");                 // add new element to the end
dogs.shift();                           // remove first element
dogs.unshift("Chihuahua");              // add new element to the beginning
dogs.splice(2, 0, "Pug", "Boxer");      // add elements (where, num to remove, element list)
var animals = dogs.concat(cats,birds);  // join two arrays (dogs followed by cats and birds)
dogs.slice(1,4);                        // elements from [1] to [4-1]
dogs.sort();                            // sort string alphabetically
dogs.reverse();                         // sort string in descending order
dogs.indexOf("Beagle");                 // return index location of Beagle: 1
dogs.lastIndexOf("Beagle");             // return last index of occurance

dogs.forEach((v, i, arr) => {           // loop over the array
    console.log("index",i,"element value",v,"entire array",arr);
});

let DOGS = dogs.map((v,i) => {          // Perform function on every value
    console.log("index",i,"element value",v);
    return v.toUpperCase();
});
```

Dates & times

```
// Dates/times - Declaring
let now = new Date();               // Current date/time
let ts = Number(new Date());        // milliseconds since 1/1/1970

// Declaring particular dates/times
Date("2021-07-29");                     // date declaration yyyy-mm-dd
Date("2021");                           // 1st January
Date("2021-07-29T19:20:00+08:00");      // yyyy-mm-ddThh:mm:ss+hh:mm (example UTC+8:00)
Date("July 29 2021");                   // long date format
Date("Jul 29 2021 19:20:00 UTC+0800"); // long form with timezone

// Dates/times - Getting parts
/* Warnings:
  .getMonth() and .setMonth() use 0-11
  .getTime() and .setTime() work in milliseconds so /1000 for use with other languages
*/
let d = new Date();
ts = d.getTime() / 1000;  // milliseconds since 1970
a = d.getDay(); getDate(); getDay(); getFullYear(); getHours(); getMinutes(); getMonth();
getSeconds();

// Setting parts of a date/time
d.setDate(); d.setFullYear(); d.setHours(); d.setMinutes(); d.setMonth(); d.setSeconds();
setTime();

// A day is internally 1.0, so you can change dates like this...
d.setDate(d.getDate() + 7); // adds a week to a date
```

HTML DOM functions

```
// Get html element with id="save"
let element = document.querySelector("#save");
// Get array of all html elements with class="item"
let elements = document.querySelectorAll(".item");
```

```
// Work with HTML element attributes
let attr = element.getAttribute( attribute_name );
let result = element.hasAttribute( attribute_name );
element.setAttribute( attribute_name, new_value );
element.removeAttribute( attribute_name );
```

```
// Event listeners
element.addEventListener( event_code, function_to_call ); // syntax
document.querySelector("#person").addEventListener( "click", hello ); // example
element.removeEventListener( event_code, function_to_call );
```

Common input events:

change - When user commits a value change
input - When the user changes the value (each character typed)

Common general events:

click, load, focus, submit

Common user interface events:

keydown, keyup, keypress
- event.key and event.keyCode: info about the key

mouseover, mousemove, mousedown, mouseup
- event.pos: the x,y coordinators
- event.button: button number

File upload example

```
<form>
    <label for="fileupload">Select file to upload</label>
    <input type="file" name="fileupload">
    <input name="uploadbutton" value="Upload">
</form>
```

```
function uploadFile(e) {
    let input = document.querySelector('input[name="fileupload"]')
    let form = new FormData();          // Create virtual form
    form.append('file', input.files[0]) // Add the file to the form
    form.append('user', 'pbaumgarten'); // Add any other fields required
    fetch('/upload', {method: 'POST', body: form }).then(function(response) {
        response.json().then(function(data) {
            // data contains the json object reply
            console.log(data);
        });
    });
}
// Attach event handler to the button click
document.querySelector("input[name='uploadbutton']").addEventListener('click', uploadFile);
```

Full example of a virtual form

```
<div class="messagebox">
    <input type="text" name="messagetext" id="messagetext" placeholder="Your message...">
    <input type="button" name="messagebutton" id="messagebutton" value="Send">
</div>
```

```
// Example that creates a virtual form, sends it to a webserver, and receives a JSON reply.
function sendMessage(e) {
    // Create a virtual HTML form
    let form = new FormData();
    form.append("message", document.querySelector("#messagetext").value)
    let fetchParameters = {
        method : 'post',
        body : form // attach the HTML form object
    };
    fetch("/newmessage", fetchParameters).then(function(response) {
        response.json().then(function(data) {
            // data will be a json object of the reply
            console.log(data);
        })
    }).catch(function(e){
        console.log("ERROR");
        console.log(e);
    });
    // Clear the textbox and put the cursor back in it
    document.querySelector('#messagetext').value = '';
    document.querySelector('#messagetext').focus();
}
function checkMessageForEnter(e) {
    // Run sendMessage() if enter key pressed
    if(e.keyCode == 13){ sendMessage(null); }
}
function main() {
    // Send message when button clicked
    document.querySelector("#messagebutton").addEventListener("click", sendMessage);
    // Also, send message when enter key pressed
    document.querySelector('#messagetext').addEventListener('keydown', checkMessageForEnter);
}
// Once everything has finished loading, run the main() function
window.onload=main;
```

Load image via fetch() and display in HTML

```
<input type="button" name="fetch" value="Fetch!">
<img id="target">
```

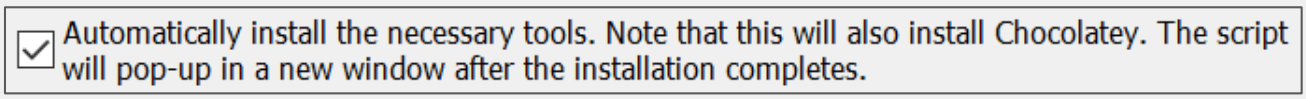
```
function getImage(e) {
    let target = document.querySelector("img#target");
    fetch('https://cataas.com/cat').then(function(response) {
        response.blob().then(function(data) { // data contains the blob object reply
            console.log("Received data blob: ",data.size,"bytes");
            var reader = new FileReader();  // Read blob as a file
            reader.onload = function(){     // Once fully read this will execute
                target.src = this.result;   // `this.result` contains a base64 data URI
            };
            reader.readAsDataURL(data);     // Start the reading process
        });
    });
}
// Add event listener to button
document.querySelector("input[name='fetch']").addEventListener("click", getImage);
```

React cheat sheet!

July 2021

Install

1. Install Node.js from https://nodejs.org/en/download/ (Windows 64bit installer)



2. create-react-app will create a folder containing a template project. From a console:

```
> npx create-react-app my-app
> cd my-app
> npm install react-router-dom axios
> npm start
```

3. Open http://localhost:3000/ to view it in the browser. It will automatically refresh as you save changes to the project.

4. When you are ready to deploy, return to the console:

```
> npm run build
> npx glup
```

Copy the `src/index.html` file to your webserver

You can experiment with React without installing using https://codesandbox.io/

Basic layout

```jsx
import logo from './logo.svg';
import './App.css';

/* Create a new component called <Hello> */
function Hello(props) {
  console.log("Hello");
  console.log(props);
  // JSX must be wrapped in one enclosing tag,
  // so can't just do <h1></h1><p></p> here
  return (
    <div>
      {/* props.children are all inner elements */}
      <h1>Hello, {props.children}!</h1>
      <p>{props.tag}</p>
    </div>
  )
}

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <Hello tag="Reacting like a boss 😆">Paul</Hello>
      </header>
    </div>
  );
}

export default App;
```

Class v functional components

React has two different syntax that can be used interchangeably for creating components. The functional method is the newer one and, in theory, can do everything the class approach can do though excess use of hooks can make it become unwieldly at times. My current preference is to use functions for “simple” components and classes for more complex components, particularly where state involves nested arrays/objects.

```
import { useEffect } from 'react';

// Functional notation
function MyButton(props) {

  // Called after first mount
  useEffect(()=>{
    console.log("did mount");
  }, []);

  // Called after change in props
  useEffect(()=>{
    console.log("Properties are now: ",props);
  }, [props]);

  return (
    <button onClick={() => props.action(props.message)}>{props.label}</button>
    )
}
```

```
import React from 'react’;

// Class notation
class MyButton extends React.Component {
  constructor( props ) {
    super(props);
  }

  // called after first mount
  componentDidMount() {
    console.log("did mount");
  }

  // called when this.props or this.state changes, prior to calling render() again
  componentDidUpdate() {
    console.log("Properties are now: ",this.props);
  }

  render() {
    return (
      <button onClick={() => this.props.action(this.props.message)}>{this.props.label}</button>
    )
  }
}
```

Either of the above will exhibit the same behavior and can be called via the following JSX:

```
{/* Note this passes the alert() function as via the action property */}
<MyButton message="You clicked the button" action={alert} label="Click me“ />
```

Form inputs

Form input requires you to keep track of the value of each input via state variables.

```
function Textbox({default, onSave}) {
  const [name, setName] = useState(default);

  return(
    <p>What is your name?</p>
    <input type="text" value={name} onChange={(e)=>setName(e.target.value)}/>
    <button onClick={()=>{onSave(name)}}>Submit</button>
  )
}
```

```
class Textbox extends React.Component {
  constructor(props) {
    super(props);
    this.state.name = this.props.default;
  }

  render() {
    return(
      <p>What is your name?</p>
      <input type="text" value={this.state.name}
         onChange={(e)=>this.setState({name: e.target.value})}/>
      <button onClick={()=>{this.props.onSave(this.state.name)}}>Submit</button>
    )
  }
}
```

```
{/* Both of these could be called like this */}
<Textbox default="" onSave={alert}/>
```

For Class component functions to have access to instance variables such as state etc, they must be called within render() using

`() => this.functionName()`
... instead of ...
`this.functionName`

```
// Example for a <select> list
function Pick({initial, callBack, items}) {
  const [day, setDay] = useState(initial);
  return(
    <div>
    <select value={day} onChange={(e)=>setDay(e.target.value)}>
      { items.map(value => {
        return(<option key={value} value={value}>{value}</option>)
      })}
    </select>
    <button onClick={()=>callBack(day)}>Set</button>
    </div>
  )
}

const list = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"];

export default function App() {
  const [day, setDay] = useState('');
  return (
    <div className="App">
      <p>What is your favourite day?</p>
      <Pick initial="Saturday" items={list} callBack={(favourite)=>{setDay(favourite)}}/>
      <p>Your favourite day is {day}</p>
    </div>
  );
}
```

Routing with react-router-dom

```
import './App.css';
import Welcome from './Welcome';
import Timer from './Timer';
import Counter from './Counter';
import { BrowserRouter as Router, Route, Switch, Link, Redirect, useParams } from "react-
router-dom";

function Page1() {
  return (
    <div>
      <Welcome name="Mr Baumgarten"></Welcome>
      <Timer></Timer>
      <Counter></Counter>
    </div>
  )
}

function Page2() {
  return(<h2>Page 2</h2>)
}

function Page3() {
  let { userid } = useParams();
  return(<h2>Page3 with param {userid}</h2>)
}

function SideBar() {
  return (
    <div>
      <ul>
        <li><Link to="/">Home</Link></li>
        <li><Link to="/page1">Page 1</Link></li>
        <li><Link to="/page2">Page 2</Link></li>
        <li><Link to="/page3/pbaum">Page 3 for pbaum</Link></li>
        <li><Link to="/page3/jdoe">Page 3 for jdoe</Link></li>
      </ul>
    </div>
  )
}

export default function App() {
  return (
    <Router>
      <SideBar/>
      <Switch> {/* Will search through routes and render first that matches */}
        <Route exact path="/">
          <Redirect to="/page1" />
        </Route>
        <Route exact path="/page1">
          <Page1/>
        </Route>
        <Route exact path="/page2">
          <Page2/>
        </Route>
        <Route exact path="/page3/:userid">
          <Page3/>
        </Route>
      </Switch>
    </Router>
  );
}
```

Docs for routing at
https://reactrouter.com/web/guides/quick-start

```
import { withRouter } from 'react-router-dom';

class Page3 extends React.Component {
  constructor(props) {
    super(props);
  }

  render() {
    // routing parameters in
    // this.props.match.params...
    return(<h2>Page 3 with param
      {this.props.match.params.userid}</h2>)
  }
}

// withRouter() will supply the params to props
export default withRouter(YourComponent);
```

Getting route parameters with function hooks use `useParams()` whereas classes use `this.props.match.params`

Rendering lists

One common approach is use .map() to iterate over arrays to custom render lists of components.

```
import React from "react";
import {
  BrowserRouter as Router, Switch, Route, Link
} from "react-router-dom";

function Nav(props) {
  // Note: Each child in a list should have a unique `key` prop so react track changes
  return (
    <nav>
      <ul>
        { props.links.map( (element,i) => {
          return (
            <li key={i}><Link to={element.to}>{element.label}</Link></li>
        )})}
      </ul>
    </nav>
  )
}

function App() {
  const opts = [
    {to: "/", label: "Home"},
    {to: "/welcome", label: "Welcome"},
    {to: "/about", label: "About"},
    {to: "/contact", label: "Contact"},
    {to: "/help", label: "Help"}
  ]
  return (
    <div className="App">
      <Router>
        <div>
          <Nav links={opts}></Nav>
          <Switch>
            <Route path="/welcome"><Welcome /></Route>
            <Route path="/about"><About /></Route>
            <Route path="/contact"><Contact /></Route>
            <Route><Redirect to="/welcome"/></Route>
          </Switch>
        </div>
      </Router>
    </div>
  );
}

export default App;
```

Fetching content with axios

```
import React, { useState, useEffect } from 'react';
import axios from "axios";

async function getWeatherForecast() {
    let url = "https://data.weather.gov.hk/weatherAPI/opendata/weather.php"
    let parameters = {
        "dataType": "fnd",
        "lang": "en"
    }
    return await axios.get(url, {params: parameters})
    .then((res) => {
      console.log("Received weather");
      return res;
    })
    .catch((err) => {
      return err;
    });
}

export default function AxiosDemo() {
    const [forecast, setForecast] = useState();

    useEffect(() => {
        // Initiate axios to obtain data
        getWeatherForecast().then((res) => {
            // When axios reply received, save it to state
            // (remember function will auto re-render when states change)
            setForecast(res.data)
        });
        console.log("Requested weather");
    }, []); // Execute on first render

    // If axios hasn't returned the data yet, display a waiting screen
    if (! forecast) {
        return(
        <h1>Fetching HK weather forecast...</h1>
        ) // Remember return() will exit the function
    }

    // If still here, we have data
    let tomorrow = forecast.weatherForecast[0];
    return(
        <div>
            <h1>HK weather forecast</h1>
            <p>{ forecast.generalSituation }</p>
            <p>For { tomorrow.week }:
               Maximum { tomorrow.forecastMaxtemp.value },
               minimum { tomorrow.forecastMintemp.value }
            </p>
            <p>{ tomorrow.forecastWeather }</p>
        </div>
    )
}
```

`async` indicates this function will have a time delay before returning, so the javascript that calls it should carry on without waiting for the response. Notice in the console the "Requested weather" will print before the "Received weather" indicating the code continued on without waiting for `getWeatherForecast()` to finish.

Docs for axios at https://github.com/axios/axios

While it is very possible to use fetch(), a lot of react examples/tutorials tend to use the axios library so here is a quick demo. For discussion of fetch() vs axios see https://stackoverflow.com/a/50326744

SQLite cheat sheet!

July 2021

Creating a table

```
CREATE TABLE 'person' (
    'id'            INTEGER AUTOINCREMENT,
    'givenName'     TEXT,
    'familyName'    TEXT,
    'email'         TEXT NOT NULL,
    PRIMARY KEY('id')
);
```

Insert / update / delete

```
-- Insert a record
INSERT INTO table (field, field, ...) VALUES (value, value, ...);

-- Update one or more records
UPDATE table SET field=value, field=value WHERE field=value;

-- Delete one or more records
DELETE FROM table WHERE field=value;
```

Querying data

```
-- Get every field of every record from a table.
SELECT * FROM table;

-- Get some fields for every record from a table.
SELECT field, field, field FROM table;

-- Get records that match where field is set to value.
SELECT … FROM table WHERE field = value;

-- Get records sorted by field
SELECT * FROM table WHERE field=value ORDER BY field2;

-- Get first 10 records
SELECT * FROM table WHERE field=value ORDER BY field2 LIMIT 10;

-- Get records that match where two fields have set values.
SELECT … FROM table WHERE (field = value) AND (field = value);

-- Get records that match where two possible field/values exist.
SELECT … FROM table WHERE (field = value) OR (field = value);

-- Get records from two tables
SELECT * FROM (table1 LEFT JOIN table2 ON table1.field=table2.field) WHERE table1.field=value;
```