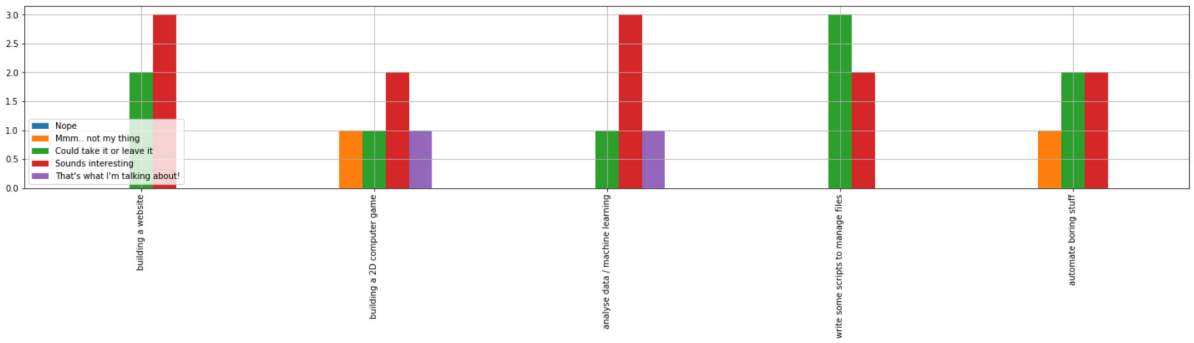
Survey Results

4 responses

- **Mode of communication**: Unanimously Email
- **Programming proficiency**: Mostly beginners (expected)
- **after-lesson commitment**: very diverse answers
- **Preferred Learning styles**: Demonstration & Collaboration
- **Interests**: ...

In [4]:
```
# DON'T WORRY IF YOU DON'T UNDERSTAND THIS CODE JUST YET
# IT'S ONLY TO PLOT THE RESULT OF THE SURVEY

import pandas as pd
%matplotlib inline

df = pd.DataFrame({
    'Nope':                          [0,0,0,0,0],
    'Mmm.. not my thing':            [0,1,0,0,1],
    'Could take it or leave it':     [2,1,1,3,2],
    'Sounds interesting':            [3,2,3,2,2],
    "That's what I'm talking about!": [0,1,1,0,0],
}, index=['building a website', 'building a 2D computer game', 'analyse data / mach
ine learning', 'write some scripts to manage files', 'automate boring stuff'])

df.plot.bar(figsize=(25,4), grid=True).legend(loc=3)

weights = [-2,-1,0,1,2]
activity_scores = (df*weights).sum(axis=1)

# RESULT: data analysis for the WIN! :)
```

Out[4]: <matplotlib.legend.Legend at 0x249b1b39808>



Home Work (NO CLASS ON THE 23RD!)

During these 2 weeks, find dataset(s) that interest you, so we can analyse them.

Don't know where to start?

- https://www.quora.com/What-are-some-interesting-data-sets-available-out-there (https://www.quora.com/What-are-some-interesting-data-sets-available-out-there)
- https://statistics.gov.scot/home (https://statistics.gov.scot/home)
- https://www.dataquest.io/blog/free-datasets-for-projects/ (https://www.dataquest.io/blog/free-datasets-for-projects/)
- .. or something you work with (and is anonymised enough)

Basics Overview continues..

- **Aggregate Types**
  - lists (sequence of items)
  - tuples (immutable lists)
  - dictionaries (key-value pairs)
- **Flow Control**
  - if.. elif.. else.. (Conditionals)
  - for i in seq (For Loop)
  - while i > 0 (While Loop)
- **Functions**
  - def myfunc(a, b, c=0)
- **Classes**
  - class MyClass

Lists

editable (mutable) sequence of items

In [2]:
```
my_wierd_list = ['Jim', 3, False, [1,2]]  # each element can be a different type

my_list = ['John', 'Paul', 'George', 'Ringo']  # most common is all the same type
```

In [8]:
```
# get first item
my_list[0]
```

Out[8]: 'John'

In [9]:
```
#get last item
my_list[-1]
```

Out[9]: 'Ringo'

In [10]:
```
# get second item?
my_list[1]
```

Out[10]: 'Paul'

In [11]:
```
# get second-to-last item?
my_list[-2]
```

Out[11]:
```
'George'
```

List slicing

also used for **arrays**, **matrices**, **pd.Series** and **strings**

LIST[start:end:step]

slicing

In [12]:
```
# taking first two
my_list[:2]    # same as my_list[0:2:1]
```

Out[12]:
```
['John', 'Paul']
```

In [13]:
```
# taking last two
my_list[-2:]    # same as my_list[-2::1]
```

Out[13]:
```
['George', 'Ringo']
```

In [16]:
```
# taking middle two?
my_list[1:-1]  # or my_list[1:3]
```

Out[16]:
```
['Paul', 'George']
```

In [3]:
```
# taking one every two (even numbers)
my_list[::2]
```

Out[3]:
```
['John', 'George']
```

In [20]:
```
# reversing the order?
my_list[::-1]
```

Out[20]:
```
['Ringo', 'George', 'Paul', 'John']
```

In [21]:
```
# sorting a list
sorted(my_list)
```

Out[21]:
```
['George', 'John', 'Paul', 'Ringo']
```

In [25]:
```
my_list
```

Out[25]:
```
['John', 'Paul', 'THE George', 'Ringo']
```

In [24]:
```
# changing one value
print('before: ', my_list)
my_list[2] = 'THE George'
print('after: ', my_list)
```

```
before:  ['John', 'Paul', 'George', 'Ringo']
after:  ['John', 'Paul', 'THE George', 'Ringo']
```

In [26]:
```
# adding elements to a list
my_list.append('Pete')

my_list
```

Out[26]:
```
['John', 'Paul', 'THE George', 'Ringo', 'Pete']
```

In [28]:
```
# checking if element in list
'Pete' in my_list
```

Out[28]:
```
False
```

Tuples (very briefly)

basically lists that cannot be edited (immutable)

In [29]:
```
my_tuple = ('John', 'Paul', 'George', 'Ringo')  # most common is all the same type

sorted(my_tuple[-2:])
```

Out[29]:
```
['George', 'Ringo']
```

In [30]:
```
my_tuple.append(42)  # won't work
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-30-1633a8bf04b2> in <module>
----> 1 my_tuple.append(42)  # won't work

AttributeError: 'tuple' object has no attribute 'append'
```

In [31]:
```
my_tuple[2] = 'THE George'  # won't work
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-31-cf84bf8e4fd8> in <module>
----> 1 my_tuple[2] = 'THE George'  # won't work

TypeError: 'tuple' object does not support item assignment
```

Dictionaries

Collection of key-value pairs

In [32]:
```
the_beatles_instruments = {
    'John': ['vocals', 'rhythm and lead guitar', 'keyboards', 'harmonica', 'bass gu
itar'],
    'Paul':  ['vocals', 'bass guitar', 'rhythm and lead guitar', 'keyboards', 'drum
s'],
    'George': ['lead and rhythm guitar', 'vocals', 'sitar', 'keyboards', 'bass guit
ar'],
    'Ringo': ['drums', 'percussion', 'vocals'],
}

the_beatles_tenure = {
    'John': {'start': 1960, 'end': 1969},
    'Paul':  [1960, 1970],
    'George': (1960, 1970),
    'Ringo': '1962-1970',
}
```

In [33]:
```
# accessing a value
the_beatles_instruments['Paul']
```

Out[33]:
```
['vocals', 'bass guitar', 'rhythm and lead guitar', 'keyboards', 'drums']
```

In [36]:
```
# starting year of John?
the_beatles_tenure['John']['start']
```

Out[36]:
```
1960
```

In [38]:
```
# adding a new value
the_beatles_instruments['Pete'] = ['drums']

the_beatles_instruments
```

Out[38]:
```
{'John': ['vocals',
  'rhythm and lead guitar',
  'keyboards',
  'harmonica',
  'bass guitar'],
 'Paul': ['vocals',
  'bass guitar',
  'rhythm and lead guitar',
  'keyboards',
  'drums'],
 'George': ['lead and rhythm guitar',
  'vocals',
  'sitar',
  'keyboards',
  'bass guitar'],
 'Ringo': ['drums', 'percussion', 'vocals'],
 'Pete': ['drums']}
```

In [39]:
```
# getting all keys
the_beatles_tenure.keys()
```

Out[39]:
```
dict_keys(['John', 'Paul', 'George', 'Ringo'])
```

In [40]:
```
# getting all values
the_beatles_tenure.values()
```

Out[40]:
```
dict_values([{'start': 1960, 'end': 1969}, [1960, 1970], (1960, 1970), '1962-197
0'])
```

In [41]:
```
# getting all keys-values as pairs
the_beatles_tenure.items()
```

Out[41]:
```
dict_items([('John', {'start': 1960, 'end': 1969}), ('Paul', [1960, 1970]), ('Ge
orge', (1960, 1970)), ('Ringo', '1962-1970')])
```

If Statement

diverting the execution flow depending on 1+ condition

In [42]:
```
happy = True
know_it = True
```

In [50]:
```
if happy and know_it:
    print('hands go clap clap')
elif not know_it:
    print('hands go clap')
else:
    print('...silence...')
```

```
hands go clap
```

In [48]:
```
# what result do I expect with.. ?
happy = True
know_it = False
```

In [46]:
```
# what result do I expect with.. ?
happy = False
know_it = True
```

In [ ]:
```
# what result do I expect with.. ?
happy = False
know_it = False
```

For Loop

runs a part of the code multiple times: one for every item in the sequence

In [51]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

In [52]:
```
print(the_beatles_instruments)
```

```
{'John': ['vocals', 'rhythm and lead guitar', 'keyboards', 'harmonica', 'bass gu
itar'], 'Paul': ['vocals', 'bass guitar', 'rhythm and lead guitar', 'keyboards',
'drums'], 'George': ['lead and rhythm guitar', 'vocals', 'sitar', 'keyboards', '
bass guitar'], 'Ringo': ['drums', 'percussion', 'vocals'], 'Pete': ['drums']}
```

In [6]:
```
d = {1: 'one', 4:'four'}

for k in d.items():
    print('pair:', k)
```

```
pair: (1, 'one')
pair: (4, 'four')
key: 1 value: one
key: 4 value: four
```

In [ ]:
```
for k,v in d.items():
    print('key:', k, 'value:', v)
```

In [61]:
```
# NESTED LOOP
# I heard you like loops so I put a loop in a loop...
for member, instruments in the_beatles_instruments.items():
    for instrument in instruments:
        print(member, 'plays', instrument)
```

```
John plays vocals
John plays rhythm and lead guitar
John plays keyboards
John plays harmonica
John plays bass guitar
Paul plays vocals
Paul plays bass guitar
Paul plays rhythm and lead guitar
Paul plays keyboards
Paul plays drums
George plays lead and rhythm guitar
George plays vocals
George plays sitar
George plays keyboards
George plays bass guitar
Ringo plays drums
Ringo plays percussion
Ringo plays vocals
Pete plays drums
```

While Loop

runs a part of the code multiple times: until a condition is met

In [62]:
```
current_value = 0
while current_value <= 5:
    print('the current value is: ', current_value)
    current_value += 1
```

```
the current value is:  0
the current value is:  1
the current value is:  2
the current value is:  3
the current value is:  4
the current value is:  5
```

Function

ways to group and reuse code that can be logically isolated

In [72]:
```
# defining a function
def simple_greeting():
    print('Hi')
```

In [73]:
```
# calling a function (using it)
simple_greeting()
```

```
Hi
```

In [7]:
```
def greeting(name, lastname):
    print('Hi,', name, lastname)

greeting('Bob', 'Kelso')                  # passing argument as positional (ORDER MA
TTERS)
greeting(lastname='Kelso', name='Bob')    # passing argument as keyword (ORDER DOES
NOT MATTER)
```

```
Hi, Bob Kelso
Hi, Bob Kelso
```

In [8]:
```
def greeting_with_default(name='you'):
    print('Hi,', name)

greeting_with_default()
greeting_with_default('Bob')
greeting_with_default(name='Bob')
```

```
Hi, you
Hi, Bob
Hi, Bob
```

In [68]:
```
# returning values instead of printing it
def duplicate(x):
    return 2*x   # instead of printing it we are returing it..

douplex = duplicate(3)  # ... so the result can be assigned to a function.
douplex
```

Out[68]:
```
6
```

In [69]:
```
# what happens if I do.. ?
duplicate(duplicate(3))
```

Out[69]:
```
12
```

Classes

Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.

In [70]:
```python
# defining the class
class Dog:
    def __init__(self, name):
        self.name = name
        self.tricks = []

    def add_trick(self, trick):
        self.tricks.append(trick)

# Don't worry if you don't fully understand the above, focus on the below :)

# creating and using an instance of it
d = Dog('Fido')
d.add_trick('roll over')
d.add_trick('play dead')
d.tricks
```

Out[70]:
```
['roll over', 'play dead']
```

`Dog` is the **class**, which is a **blueprint** to create object that you can actually use

`d` is the **instance (object)** that you can use to do whatever you need

Revisited High-Low

using what we have learned so far..

`input(message)` is a built-in python function that requests user input showing a message

In [ ]:
```
import random

CARDS = range(1, 13+1)
card = random.choice(CARDS)

def ask_guess(card):
    print('====================')
    print('The current card is', card)
    guess = input('Is the next one going to be higher ("h") or lower ("l")? ("q" to
quit)')
    return guess

def evaluate_result(card, new_card, guess):
    if new_card == card:
        print('They are exactly the same card, you lose :(')
    elif (new_card > card and guess == 'h') or (new_card < card and guess == 'l'):
        print('Well done! you guessed correcty :)')
    elif (new_card > card and guess == 'l') or (new_card < card and guess == 'h'):
        print('Wrong :(')
    elif guess not in ['h','l','q']:
        print('you dummy! you can only enter "h", "l" or "q"')
    else:
        print('?? how did you even get here ??')


guess = ask_guess(card)
while guess != 'q':
    new_card = random.choice(CARDS)
    print('The new card is: ', new_card)
    evaluate_result(card, new_card, guess)
    card = new_card
    guess = ask_guess(card)

print('====================')
print("Thanks for playing :)")
```

```
====================
The current card is 11

The new card is:  4
<  >
you dummy!
====================
The current card is 4
```

Exercises

mostrly taken from [codewars.com (https://www.codewars.com)](https://www.codewars.com)

In [ ]:
```
# Write a program that prints the numbers from 1 to 20. But for multiples of three
prints “Fizz” instead of the number and for the multiples of five print “Buzz”. For
numbers which are multiples of both three and five print “FizzBuzz”.
for n in range(1, 20+1):
    ...
    print(...)

# hint: the first prints should be: 1 2 Fizz 4 Buzz Fizz ...
```

In [ ]:
```
#Write a function called repeatString which repeats the given String src exactly co
unt times.
def repeatStr(n, s):
    ...
    return result

# Examples:
# repeatStr(6, "I")       >>    "IIIIII"
# repeatStr(5, "Hello")   >>   "HelloHelloHelloHelloHello"
```

In [ ]:
```
# Create a function that removes the first and last characters of a string.
def trim(x):
    ...
    return trimmed_x

# Examples:
# trim("Chocolate")   >>   "hocolat"
# trim("L4W")         >>   "4"
```

In [ ]:
```
# you are given a number and have to make it negative. But maybe the number is alre
ady negative?
def make_negative(x):
    ...
    return result

# Examples:
# make_negative(1)    >> -1
# make_negative(-5)   >> -5
# make_negative(0)    >> 0
```

In [ ]:
```
# Given an list of integers your solution should find the smallest integer.
def find_min(x):
    ...
    return result

# For example:
# Given [34, 15, 88, 2] your solution will return 2
# Given [34, -345, -1, 100] your solution will return -345
```

In [ ]:
```
# Count the Trues in a list
def count_trues(sequence):
    ...
    return num_of_true

# For example:
# count_trues([True,  True,  False, True,  False,  True]  >>  4
```

In [ ]:
```
# Summation
#Write a function that finds the summation of every number from 1 to num. The numbe
r will always be a positive integer greater than 0.
def summation(x):
    ...
    return result


# For example:
#summation(2) -> 3
# 1 + 2
# summation(8) -> 36
# 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8
```