

Name: Khushi Nitinkumar Patel

PRN: 2020BTECS00037

Batch: T2

Assignment No 7

1. Create a `Spaceship` class with three variable properties: `name`, `health`, and `position`. The default value of `name` should be an empty string and `health` should be 0. `position` will be represented by an `Int` where negative numbers place the ship further to the left and positive numbers place the ship further to the right. The default value of `position` should be 0.

```
1 //1
2 class Spaceship {
3     var name: String
4     var health: Int
5     var position: Int
6
7     init(name: String = "", health: Int = 0, position:
8         Int = 0) {
9         self.name = name
10        self.health = health
11        self.position = position
12    }
13 }
```

2. Create a `let` constant called `falcon` and assign it to an instance of `Spaceship`. After initialization, set `name` to "Falcon."

```
//2
let falcon = Spaceship()
falcon.name = "Falcon"
```

☐ Spaceship
☒ Spaceship

3. Go back and add a method called `moveLeft()` to the definition of `Spaceship`. This method should adjust the position of the spaceship to the left by one. Add a similar method called `moveRight()` that moves the spaceship to the right. Once these methods exist, use them to move `falcon` to the left twice and to the right once. Print the new position of `falcon` after each change in position.

```

1 //1
2 class Spaceship {
3     var name: String
4     var health: Int
5     var position: Int
6
7     init(name: String = "", health: Int = 0, position: Int = 0) {
8         self.name = name
9         self.health = health
10        self.position = position
11    }
12    func moveLeft() {
13        position -= 1
14        print("\(name) moved to the left. New position: \(position)")
15    }
16
17    func moveRight() {
18        position += 1
19        print("\(name) moved to the right. New position: \(position)")
20    }
21 }
22
23 //2
24 let falcon = Spaceship()
25 //3
26 falcon.name = "Falcon"
27
28 falcon.moveLeft() // prints "Falcon moved to the left. New position: -1"
29 falcon.moveLeft() // prints "Falcon moved to the left. New position: -2"
30 falcon.moveRight() // prints "Falcon moved to the right. New position: -1"
31
32

```

(2 times)

"Falcon moved to the right..."

Spaceship

Spaceship

Spaceship

Spaceship

Spaceship

Line: 10 Col: 33

```

Falcon moved to the left. New position: -1
Falcon moved to the left. New position: -2
Falcon moved to the right. New position: -1

```

- The last thing `Spaceship` needs for this example is a method to handle what happens if the ship gets hit. Go back and add a method `wasHit()` to `Spaceship` that will decrement the ship's health by 5, then if `health` is less than or equal to 0 will print "Sorry, your ship was hit one too many times. Do you want to play again?" Once this method exists, call it on `falcon` and print out the value of `health`.

```

29 //2
30 let falcon = Spaceship()
31 //3
32 falcon.name = "Falcon"
33
34 falcon.moveLeft() // prints "Falcon moved to the left. New position: -1"
35 falcon.moveLeft() // prints "Falcon moved to the left. New position: -2"
36 falcon.moveRight() // prints "Falcon moved to the right. New position: -1"
37 //4
38 falcon.wasHit()
39 print("Falcon's health is now \(falcon.health)")
40

```

Spaceship

Spaceship

Spaceship

Spaceship

Spaceship

Spaceship

"Falcon's health is now -5\n"

Line: 39 Col: 50

```

Falcon moved to the left. New position: -1
Falcon moved to the left. New position: -2
Falcon moved to the right. New position: -1
Sorry, your ship was hit one too many times. Do you want to play again?
Falcon's health is now -5

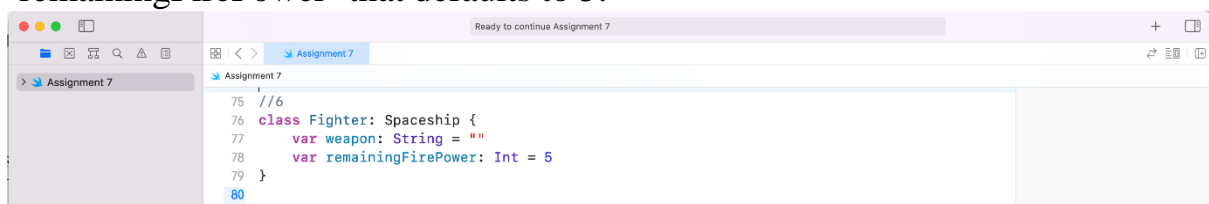
```

- Note: The exercises below are based on a game where a spaceship avoids obstacles in space. The ship is positioned at the bottom of a coordinate system

and can only move left and right while obstacles "fall" from top to bottom. Throughout the exercises, you'll create classes to represent different types of spaceships that can be used in the game. The base class `Spaceship` has been provided for you below.

```
class Spaceship {  
    var name: String = ""  
    var health = 100  
    var position = 0  
    func moveLeft() {  
        position -= 1  
    }  
    func moveRight() {  
        position += 1  
    }  
    func wasHit() {  
        health -= 5  
        if health <= 0 {  
            print("Sorry, your ship was hit one too many times. Do you want to play  
again?")  
        }  
    }  
}
```

6. Define a new class `Fighter` that inherits from `Spaceship`. Add a variable property `weapon` that defaults to an empty string and a variable property `remainingFirePower` that defaults to 5.



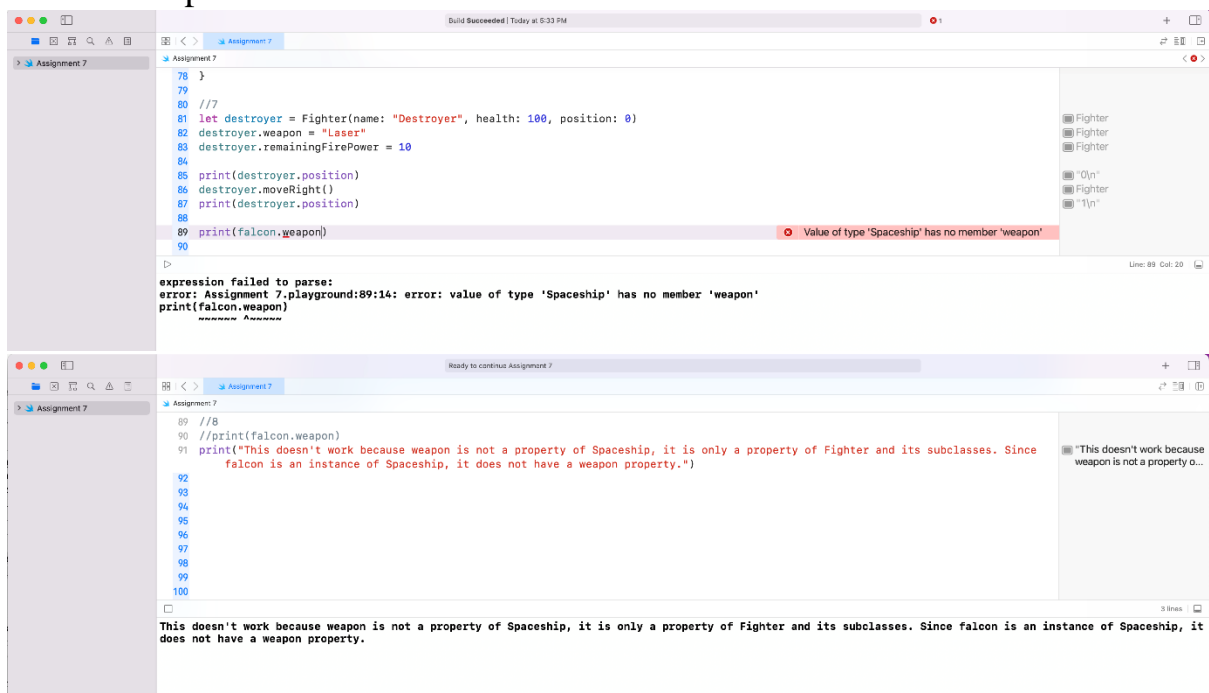
7. Create a new instance of `Fighter` called `destroyer`. A `Fighter` will be able to shoot incoming objects to avoid colliding with them. After initialization, set `weapon` to "Laser" and `remainingFirePower` to 10. Note that since `Fighter` inherits from `Spaceship`, it also has properties for `name`, `health`, and `position`, and has methods for `moveLeft()`, `moveRight()`, and `wasHit()` even though you did not specifically add them to the declaration of `Fighter`.

Knowing that, set `name` to "Destroyer," print `position`, then call `moveRight()` and print `position` again.



```
73
74 //6
75 class Fighter: Spaceship {
76     var weapon: String = ""
77     var remainingFirePower: Int = 5
78 }
79
80 //7
81 let destroyer = Fighter(name: "Destroyer", health: 100, position: 0)
82 destroyer.weapon = "Laser"
83 destroyer.remainingFirePower = 10
84
85 print(destroyer.position)
86 destroyer.moveRight()
87 print(destroyer.position)
88
89
0
Destroyer moved to the right. New position: 1
1
```

8. Try to print `weapon` on `falcon`. Why doesn't this work? Provide your answer in a comment or a print statement below, and remove any code you added that doesn't compile.



```
78 }
79
80 //7
81 let destroyer = Fighter(name: "Destroyer", health: 100, position: 0)
82 destroyer.weapon = "Laser"
83 destroyer.remainingFirePower = 10
84
85 print(destroyer.position)
86 destroyer.moveRight()
87 print(destroyer.position)
88
89 print(falcon.weapon)
90
expression failed to parse:
error: Assignment 7.playground:89:14: error: value of type 'Spaceship' has no member 'weapon'
print(falcon.weapon)
          ~~~~~
1
Value of type 'Spaceship' has no member 'weapon'
```

```
89 //8
90 //print(falcon.weapon)
91 print("This doesn't work because weapon is not a property of Spaceship, it is only a property of Fighter and its subclasses. Since
    falcon is an instance of Spaceship, it does not have a weapon property.")
92
93
94
95
96
97
98
99
100
This doesn't work because weapon is not a property of Spaceship, it is only a property of Fighter and its subclasses. Since falcon is an instance of Spaceship, it does not have a weapon property.
```

9. Add a method to `fighter` called `fire()`. This should check to see if `remainingFirePower` is greater than 0, and if so, should decrement `remainingFirePower` by one. If `remainingFirePower` is not greater than 0, print "You have no more fire power." Call `fire()` on `destroyer` a few times and print `remainingFirePower` after each method call.

Ready to continue Assignment 7

Assignment 7

```
93 //9
94 class Fighter: Spaceship {
95     var weapon: String = ""
96     var remainingFirePower: Int = 5
97
98     func fire() {
99         if remainingFirePower > 0 {
100             remainingFirePower -= 1
101         } else {
102             print("You have no more fire power.")
103         }
104     }
105 }
106
107 let destroyer = Fighter(name: "Destroyer")
108 destroyer.weapon = "Laser"
109 destroyer.remainingFirePower = 10
110
111 print(destroyer.position)
112 destroyer.moveRight()
113 print(destroyer.position)
114
115 destroyer.fire()
116 print(destroyer.remainingFirePower)
117
118 destroyer.fire()
119 print(destroyer.remainingFirePower)
120
121 destroyer.fire()
122 print(destroyer.remainingFirePower)
123
124 destroyer.fire()
125 print(destroyer.remainingFirePower)
126
127 destroyer.fire()
128 print(destroyer.remainingFirePower)
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Fighter

Fighter

Fighter

0\n

Fighter

1\n

Fighter

9\n

Fighter

8\n

Fighter

7\n

Fighter

6\n

Fighter

5\n

0

1

2

3

4

5

Destroyer moved to the right. New position: 1

10. Note: The exercises below are based on a game where a spaceship avoids obstacles in space. The ship is positioned at the bottom of a coordinate system and can only move left and right while obstacles "fall" from top to bottom. Throughout the exercises, you'll create classes to represent different types of spaceships that can be used in the game. The base class `Spaceship`` and one subclass `Fighter`` have been provided for you below.

```
class Spaceship {
    var name: String = ""
    var health = 100
    var position = 0
    func moveLeft() {
        position -= 1 }
    func moveRight() {
        position += 1 }
    func wasHit() {
        health -= 5 }
    if health <= 0 {

        print("Sorry, your ship was hit one too many times. Do you want to play again?") }

    }
}
```

```

class Fighter: Spaceship {
    var weapon = ""
    var remainingFirePower = 5
    func fire() {
        if remainingFirePower > 0 {
            remainingFirePower -= 1
        } else {
            print("You have no more fire power.")
        }
    }
}

```

11. Define a new class `ShieldedShip` that inherits from `Fighter`. Add a variable property `shieldStrength` that defaults to 25. Create a new instance of `ShieldedShip` called `defender`. Set `name` to "Defender" and `weapon` to "Cannon." Call `moveRight()` and print `position`, then call `fire()` and print `remainingFirePower`.

```

42 //11
43 class Fighter: Spaceship {
44     var weapon: String = ""
45     var remainingFirePower: Int = 5
46
47     func fire() {
48         if remainingFirePower > 0 {
49             remainingFirePower -= 1 } else {
50                 print("You have no more fire power.") }
51         }
52     }
53
54     class ShieldedShip: Fighter {
55         var shieldStrength: Int = 25
56     }
57
58     let defender = ShieldedShip()
59     defender.name = "Defender"
60     defender.weapon = "Cannon"
61     defender.moveRight()
62     print(defender.position)
63
64     defender.fire()
65     print(defender.remainingFirePower)

```

12. Go back to your declaration of `ShieldedShip` and override `wasHit()`. In the body of the method, check to see if `shieldStrength` is greater than 0. If it is, decrement `shieldStrength` by 5. Otherwise, decrement `health` by 5. Call `wasHit()` on `defender` and print `shieldStrength` and `health`.

```
//12
68 class ShieldedShip: Fighter {
69     var shieldStrength: Int = 25
70
71     override func wasHit() {
72         if shieldStrength > 0 {
73             shieldStrength -= 5
74         } else {
75             health -= 5
76             if health <= 0 {
77                 print("Sorry, your ship was hit one too many times. Do you want to play again?")
78             }
79         }
80     }
81 }
82
83 defender.wasHit()
84 print(defender.shieldStrength)
85 print(defender.health)
86
```

☐

20
0

13. When `shieldStrength` is 0, all `wasHit()` does is decrement `health` by 5. That's exactly what the implementation of `wasHit()` on `Spaceship` does! Instead of rewriting that, you can call through to the superclass implementation of `wasHit()`. Go back to your implementation of `wasHit()` on `ShieldedShip` and remove the code where you decrement `health` by 5 and replace it with a call to the superclass's implementation of the method. Call `wasHit()` on `defender`, then print `shieldStrength` and `health`.

```
88 //13
89
90 class ShieldedShip: Fighter {
91     var shieldStrength: Int = 25
92
93     override func wasHit() {
94         if shieldStrength > 0 {
95             shieldStrength -= 5
96         } else {
97             super.wasHit()
98         }
99         print("Shield Strength: \(shieldStrength), Health: \(health)")
100     }
101 }
102
103 defender.wasHit()
104 print("Shield Strength: \(defender.shieldStrength), Health: \(defender.health)")
```

☐

Shield Strength: 20, Health: 0
Shield Strength: 20, Health: 0

14. Note: The exercises below are based on a game where a spaceship avoids obstacles in space. The ship is positioned at the bottom of a coordinate system and can only move left and right while obstacles "fall" from top to bottom. The base class `Spaceship` and subclasses `Fighter` and `ShieldedShip` have been provided for you below. You will use these to complete the exercises.

```
class Spaceship {
    let name: String
    var health: Int
    var position: Int
    func moveLeft() {
        position -= 1
    }
    func moveRight() {
        position += 1
    }
    func wasHit() {
        health -= 5
        if health <= 0 {
            print("Sorry, your ship was hit one too many times. Do you
                want to play again?")
        }
    }
}

class Fighter: Spaceship {
    let weapon: String
    var remainingFirePower: Int
    func fire() {
        if remainingFirePower > 0 {
            remainingFirePower -= 1
        } else {
            print("You have no more fire power.")
        }
    }
}
```



```
}
```

```
class ShieldedShip: Fighter {  
    var shieldStrength: Int  
    override func wasHit() {  
        if shieldStrength > 0 {  
            shieldStrength -= 5  
        } else {  
            super.wasHit()  
        }  
    }  
}
```

15. Note that each class above has an error by the class declaration that says "Class has no initializers." Unlike structs, classes do not come with memberwise initializers because the standard memberwise initializers don't always play nicely with inheritance. You can get rid of the error by providing default values for everything, but it is common, and better practice, to simply write your own initializer. Go to the declaration of `Spaceship` and add an initializer that takes in an argument for each property on `Spaceship` and sets the properties accordingly.

```
1 //15  
2 //Example  
3 class Spaceship {  
4     var name: String  
5     var health: Int  
6     var position: Int  
7  
8     init(name: String = "", health: Int = 0, position: Int = 0) {  
9         self.name = name  
10        self.health = health  
11        self.position = position  
12    }  
13  
14    func moveLeft() {  
15        position -= 1  
16    }  
17  
18    func moveRight() {  
19        position += 1  
20    }  
21  
22    func wasHit() {  
23        health -= 5  
24        if health <= 0 {  
25            print("Sorry, your ship was hit one too many times. Do you want to play again?")  
26        }  
27    }  
28 }  
29  
30 let myShip = Spaceship(name: "Millennium Falcon", health: 100, position: 0)  
31
```

16. Then create an instance of `Spaceship` below called `falcon`. Use the memberwise initializer you just created. The ship's name should be "Falcon."

```
63
64 //16
65 let falcon = Spaceship(name: "Falcon", health: 100, position: 0)
66
67
```

17. Writing initializers for subclasses can get tricky. Your initializer needs to not only set the properties declared on the subclass, but also set all of the uninitialized properties on classes that it inherits from. Go to the declaration of `Fighter` and write an initializer that takes an argument for each property on `Fighter` and for each property on `Spaceship`. Set the properties accordingly. (Hint: you can call through to a superclass's initializer with `super.init` *after* you initialize all of the properties on the subclass).

```
class ShieldedShip: Fighter {
    var shieldStrength: Int

    init(name: String, health: Int, position: Int, weapon: String, remainingFirePower: Int,
        shieldStrength: Int) {
        self.shieldStrength = shieldStrength

        super.init(name: name, health: health, position: position, weapon: weapon,
            remainingFirePower: remainingFirePower)
    }

    override func wasHit() {
        if shieldStrength > 0 {
            shieldStrength -= 5
        } else {
            super.wasHit()
        }
    }
}
```

18. Then create an instance of `Fighter` below called `destroyer`. Use the memberwise initializer you just created. The ship's name should be "Destroyer."

```
//18
let destroyer = Fighter(name: "Destroyer", health: 80, position: 0, weapon: "Laser",
    remainingFirePower: 5)
```

19. Now go add an initializer to `ShieldedShip` that takes an argument for each property on `ShieldedShip`, `Fighter`, and `Spaceship`, and sets the properties accordingly. Remember that you can call through to the initializer on `Fighter` using `super.init`.

```

class ShieldedShip: Fighter {
    var shieldStrength: Int

    init(name: String, health: Int, position: Int, weapon: String, remainingFirePower: Int,
        shieldStrength: Int) {
        self.shieldStrength = shieldStrength

        super.init(name: name, health: health, position: position, weapon: weapon,
            remainingFirePower: remainingFirePower)
    }

    override func wasHit() {
        if shieldStrength > 0 {
            shieldStrength -= 5
        } else {
            super.wasHit()
        }
    }
}

```

20. Then create an instance of `ShieldedShip` below called `defender`. Use the memberwise initializer you just created. The ship's name should be "Defender."

```

//20
let defender = ShieldedShip(name: "Defender", health: 80, position: 0, weapon: "Cannon",
    remainingFirePower: 3, shieldStrength: 25)

```

21. Create a new constant named `sameShip` and set it equal to `falcon`. Print out the position of `sameShip` and `falcon`, then call `moveLeft()` on `sameShip` and print out the position of `sameShip` and `falcon` again. Did both positions change? Why? If both were structs instead of classes, would it be the same? Why or why not? Provide your answer in a comment or print statement below.

```

78
79 //21
80 let sameShip = falcon
81 print(sameShip.position)
82 print(falcon.position)
83
84 sameShip.moveLeft()
85 print(sameShip.position)
86 print(falcon.position)
87

```

0
0
-1
-1

Line: 80 Col: 21

22. Assume you are an event coordinator for a community charity event and are keeping a list of who has registered. Create a variable `registrationList` that will hold strings. It should be empty after initialization.

Ready to continue Assignment 7

Assignment 7

```

1 import UIKit
2
3 //Q22
4 var registrationList: [String] = []
5

```

23. Your friend Sara is the first to register for the event. Add her name to `registrationList` using the `append(_:)` method. Print the contents of the collection.

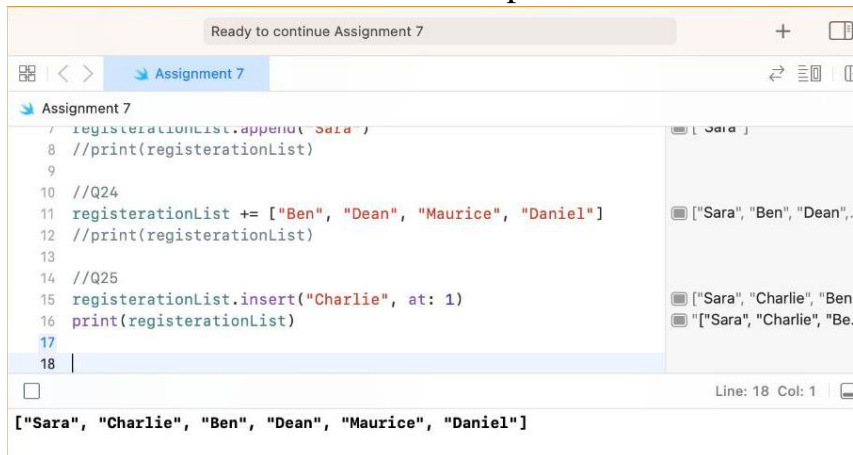


```
1 import UIKit
2
3 //Q22
4 var registrationList: [String] = []
5
6 //Q23
7 registrationList.append("Sara")
8 print(registrationList)
9
10
```

Line: 10 Col: 1

["Sara"]

24. Add four additional names into the array using the `+=` operator. All of the names should be added in one step. Print the contents of the collection.

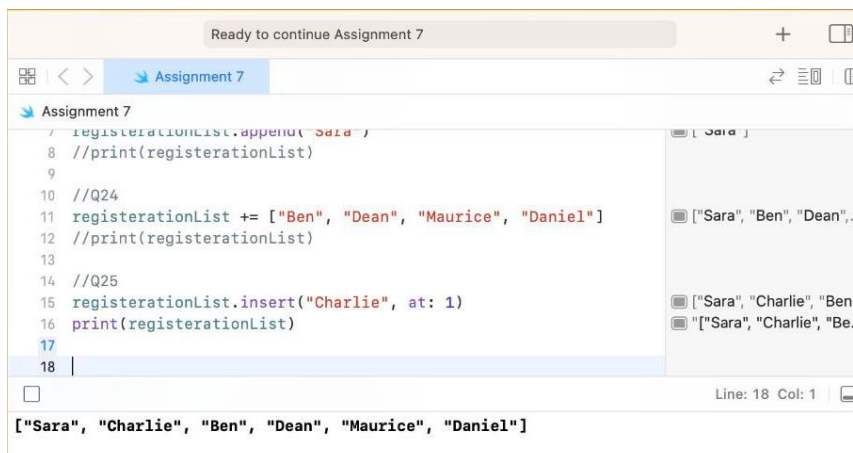


```
7 registrationList.append("Sara")
8 //print(registrationList)
9
10 //Q24
11 registrationList += ["Ben", "Dean", "Maurice", "Daniel"]
12 //print(registrationList)
13
14 //Q25
15 registrationList.insert("Charlie", at: 1)
16 print(registrationList)
17
18
```

Line: 18 Col: 1

["Sara", "Charlie", "Ben", "Dean", "Maurice", "Daniel"]

25. Use the `insert(_:at:)` method to add `Charlie` into the array as the second element. Print the contents of the collection.

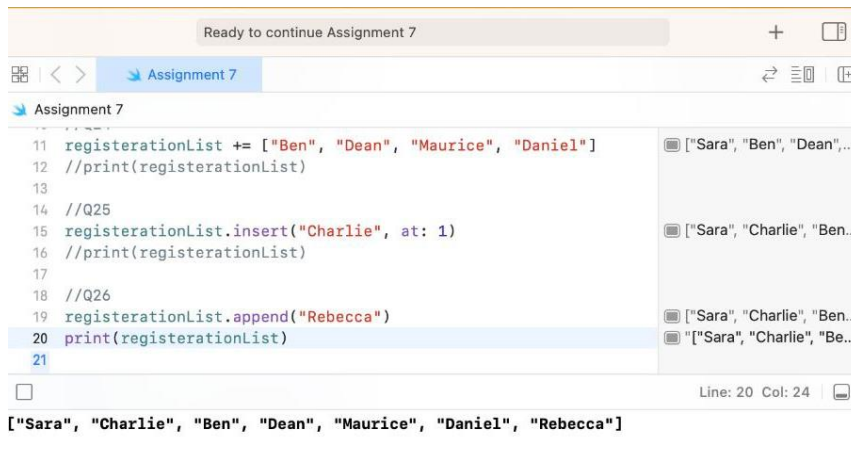


```
7 registrationList.append("Sara")
8 //print(registrationList)
9
10 //Q24
11 registrationList += ["Ben", "Dean", "Maurice", "Daniel"]
12 //print(registrationList)
13
14 //Q25
15 registrationList.insert("Charlie", at: 1)
16 print(registrationList)
17
18
```

Line: 18 Col: 1

["Sara", "Charlie", "Ben", "Dean", "Maurice", "Daniel"]

26. Somebody had a conflict and decided to transfer registration to someone else. Use array subscripting to change the sixth element to `Rebecca`. Print the contents of the collection.

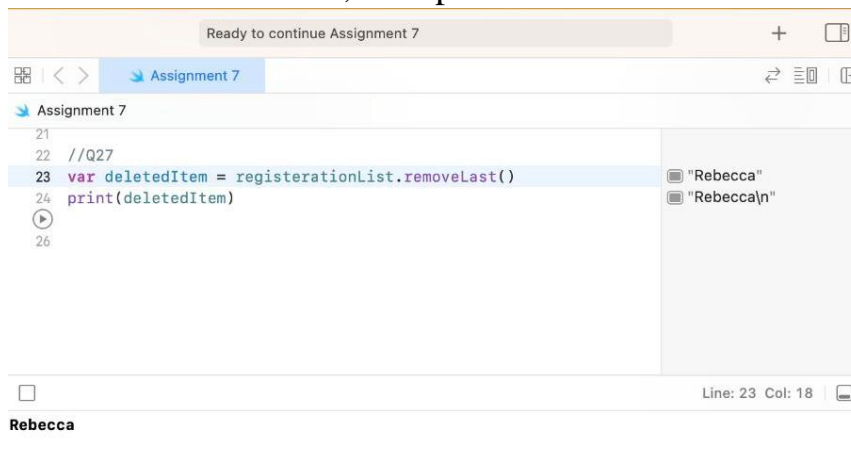


```
11 registrationList += ["Ben", "Dean", "Maurice", "Daniel"]
12 //print(registrationList)
13
14 //Q25
15 registrationList.insert("Charlie", at: 1)
16 //print(registrationList)
17
18 //Q26
19 registrationList.append("Rebecca")
20 print(registrationList)
21
```

Line: 20 Col: 24

["Sara", "Charlie", "Ben", "Dean", "Maurice", "Daniel", "Rebecca"]

27. Call `removeLast()` on `registrationList`. If done correctly, this should remove `Rebecca` from the collection. Store the result of `removeLast()` into a new constant `deletedItem`, then print `deletedItem`.



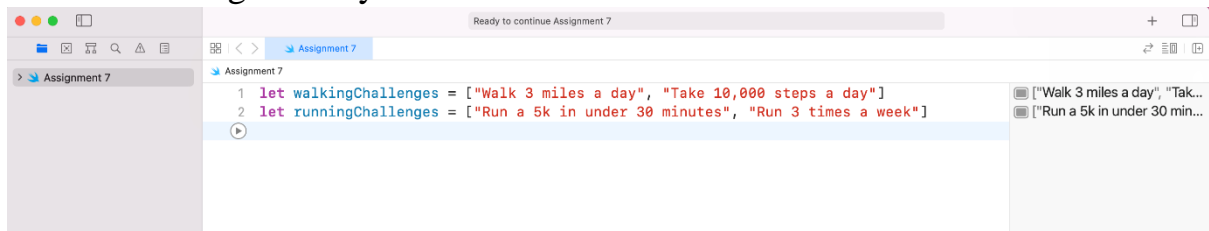
```
21
22 //Q27
23 var deletedItem = registrationList.removeLast()
24 print(deletedItem)
25
26
```

Line: 23 Col: 18

Rebecca

28. These exercises reinforce Swift concepts in the context of a fitness tracking app. Your fitness tracking app shows users a list of possible challenges, grouped by activity type (i.e. walking challenges, running challenges, calisthenics challenges, weightlifting challenges, etc.) A challenge could be as simple as "Walk 3 miles a day" or as intense as "Run 5 times a week."

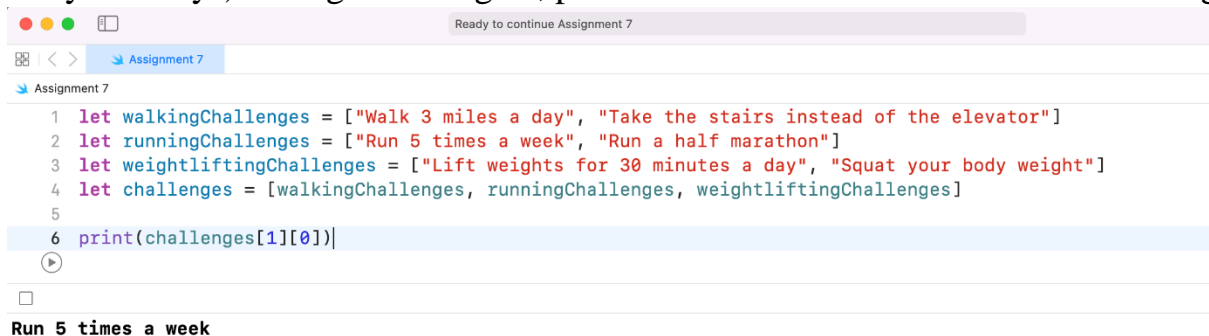
29. Using arrays of type `String`, create at least two lists, one for walking challenges, and one for running challenges. Each should have at least two challenges and should be initialized using an array literal. Feel free to create more lists for different activities.



```
1 let walkingChallenges = ["Walk 3 miles a day", "Take 10,000 steps a day"]
2 let runningChallenges = ["Run a 5k in under 30 minutes", "Run 3 times a week"]
```

["Walk 3 miles a day", "Take 10,000 steps a day"]
["Run a 5k in under 30 minutes", "Run 3 times a week"]

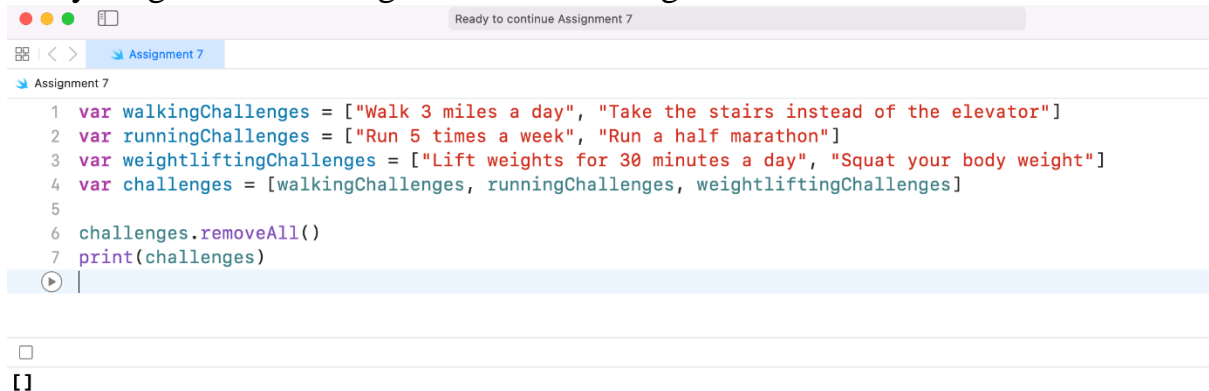
30. In your app you want to show all of these lists on the same screen grouped into sections. Create a `challenges` array that holds each of the lists you have created (it will be an array of arrays). Using `challenges`, print the first element in the second challenge list.



```
1 let walkingChallenges = ["Walk 3 miles a day", "Take the stairs instead of the elevator"]
2 let runningChallenges = ["Run 5 times a week", "Run a half marathon"]
3 let weightliftingChallenges = ["Lift weights for 30 minutes a day", "Squat your body weight"]
4 let challenges = [walkingChallenges, runningChallenges, weightliftingChallenges]
5
6 print(challenges[1][0])
```

Run 5 times a week

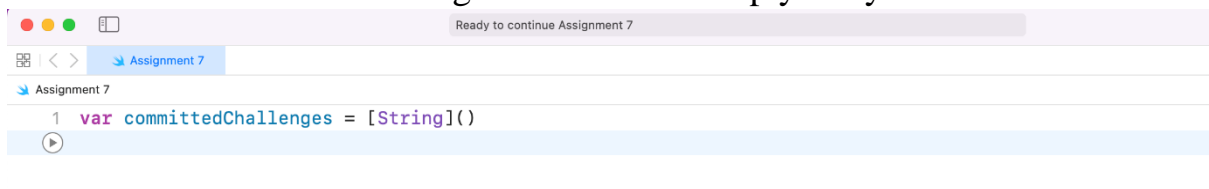
31. All of the challenges will reset at the end of the month. Use the `removeAll` to remove everything from `challenges`. Print `challenges`.



```
1 var walkingChallenges = ["Walk 3 miles a day", "Take the stairs instead of the elevator"]
2 var runningChallenges = ["Run 5 times a week", "Run a half marathon"]
3 var weightliftingChallenges = ["Lift weights for 30 minutes a day", "Squat your body weight"]
4 var challenges = [walkingChallenges, runningChallenges, weightliftingChallenges]
5
6 challenges.removeAll()
7 print(challenges)
```

[]

32. Create a new array of type `String` that will represent challenges a user has committed to instead of available challenges. It can be an empty array or have a few items in it



```
1 var committedChallenges = [String]()
```

33. Write an if statement that will use `isEmpty` to check if there is anything in the array. If there is not, print a statement asking the user to commit to a challenge. Add an else-if statement that will print "The challenge you have chosen is " if the array count is exactly 1. Then add an else statement that will print "You have chosen multiple challenges."

```
Assignment 7
1 //Q.33
2 var committedChallenges = [String]()
3 if committedChallenges.isEmpty {
4     print("Please commit to a challenge.")
5 } else if committedChallenges.count == 1 {
6     print("The challenge you have chosen is \(committedChallenges[0]).")
7 } else {
8     print("You have chosen multiple challenges.")
9 }

Please commit to a challenge.
```

34. Create a variable `[String: Int]` dictionary that can be used to look up the number of days in a particular month. Use a dictionary literal to initialize it with January, February, and March. January contains 31 days, February has 28, and March has 31. Print the dictionary.

```
Assignment 7
1 //Q.34
2 var daysInMonth = ["January": 31, "February": 28, "March": 31]
3
4 print(daysInMonth)

["February": 28, "January": 31, "March": 31]
```

35. Using subscripting syntax to add April to the collection with a value of 30. Print the dictionary.

```
Assignment 7
1 //Q.35
2 var daysInMonth = ["January": 31, "February": 28, "March": 31]
3 daysInMonth["April"] = 30
4
5 print(daysInMonth)

["April": 30, "March": 31, "January": 31, "February": 28]
```


36. It's a leap year! Update the number of days in February to 29 using the `updateValue(_:, forKey:)` method. Print the dictionary.

```
Ready to continue Assignment 7

Assignment 7

1 //Q.36
2 var daysInMonth = ["January": 31, "February": 28, "March": 31]
3
4 daysInMonth.updateValue(29, forKey: "February")
5 print(daysInMonth)
```

`["February": 29, "January": 31, "March": 31]`

37. Use if-let syntax to retrieve the number of days under "January." If the value is there, print "January has 31 days", where 31 is the value retrieved from the dictionary.

```
Ready to continue Assignment 7

Assignment 7

1 //Q.37
2 let daysInMonth = ["January": 31, "February": 29, "March": 31, "April": 30]
3
4 if let days = daysInMonth["January"] {
5     print("January has \(days) days")
6 } else {
7     print("January is not a valid key in the dictionary")
8 }
```

`January has 31 days`

38. . Given the following arrays, create a new `[String : [String]]` dictionary. `shapesArray` should use the key "Shapes" and `colorsArray` should use the key "Colors." Print the resulting dictionary.

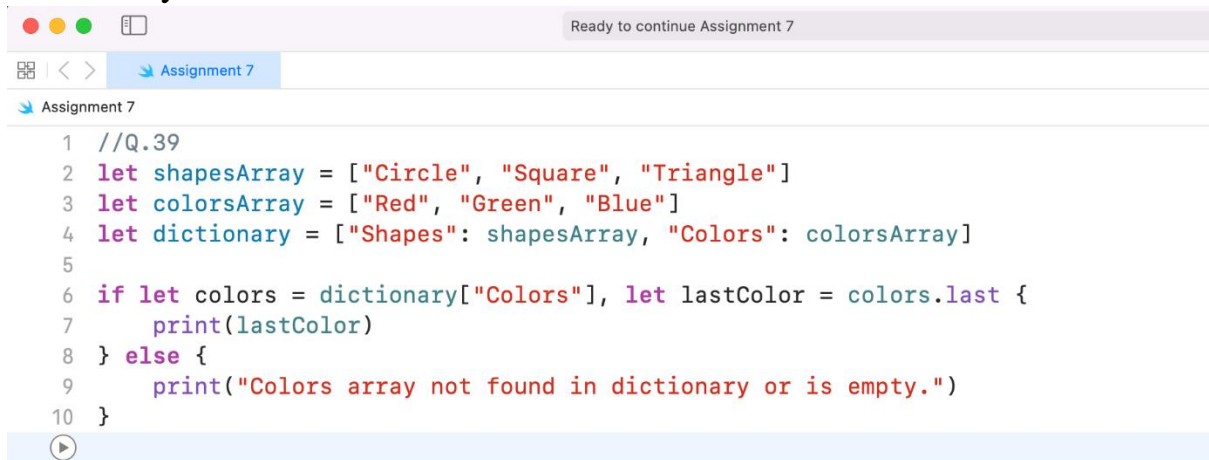
```
Ready to continue Assignment 7

Assignment 7

1 //Q.38
2 let shapesArray = ["Circle", "Square", "Triangle"]
3 let colorsArray = ["Red", "Green", "Blue"]
4
5 let dictionary = ["Shapes": shapesArray, "Colors": colorsArray]
6
7 print(dictionary)
```

`["Shapes": ["Circle", "Square", "Triangle"], "Colors": ["Red", "Green", "Blue"]]`

39. Print the last element of `colorsArray`, accessing it through the dictionary you've created. You'll have to use if-let syntax or the force unwrap operator to unwrap what is returned from the dictionary before you can access an element of the array

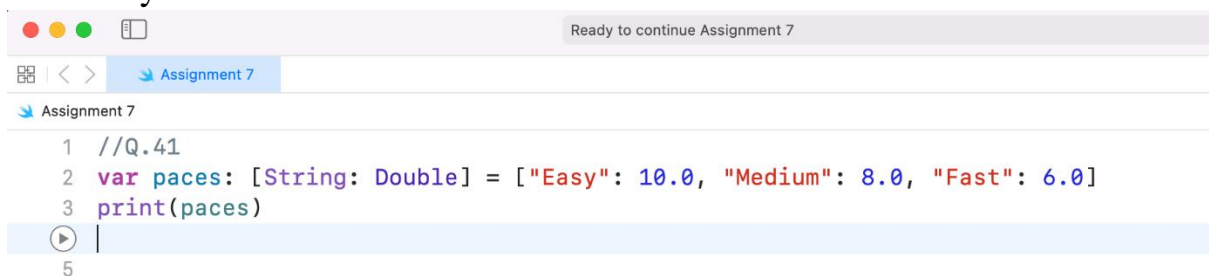


```
1 //Q.39
2 let shapesArray = ["Circle", "Square", "Triangle"]
3 let colorsArray = ["Red", "Green", "Blue"]
4 let dictionary = ["Shapes": shapesArray, "Colors": colorsArray]
5
6 if let colors = dictionary["Colors"], let lastColor = colors.last {
7     print(lastColor)
8 } else {
9     print("Colors array not found in dictionary or is empty.")
10 }
```

☐

Blue

40. These exercises reinforce Swift concepts in the context of a fitness tracking app. In previous app exercises you've written code to help users with run pacing. You decide that you could use a dictionary to let users store different paces that they regularly run at or do interval training with.
41. Create a dictionary `paces` of type [String: Double] and assign it a dictionary literal with "Easy", "Medium", and "Fast" keys corresponding to values of 10.0, 8.0, and 6.0. These numbers correspond to mile pace in minutes. Print the dictionary

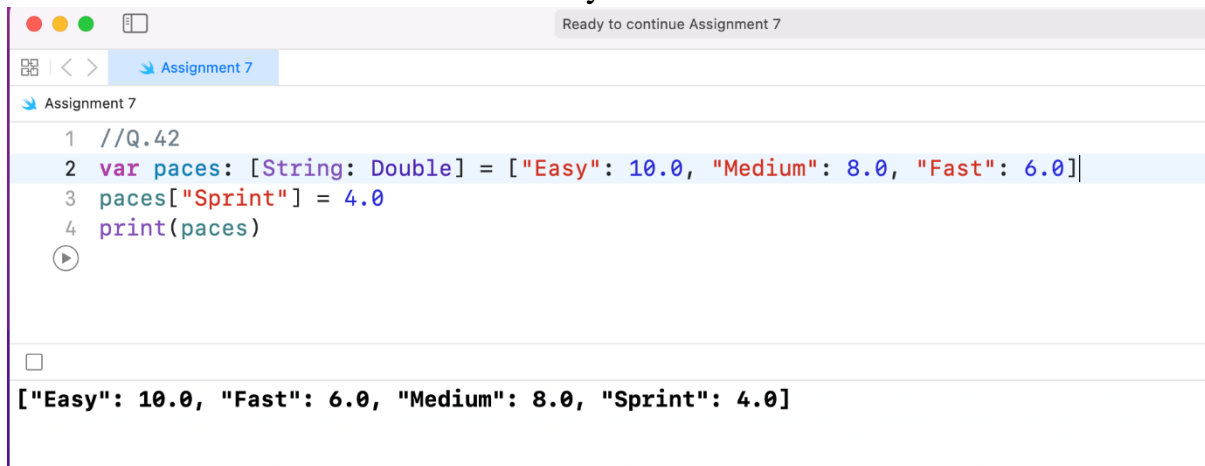


```
1 //Q.41
2 var paces: [String: Double] = ["Easy": 10.0, "Medium": 8.0, "Fast": 6.0]
3 print(paces)
4
5
```

☐

["Easy": 10.0, "Medium": 8.0, "Fast": 6.0]

42. Add a new key/value pair to the dictionary. The key should be "Sprint" and the value should be 4.0. Print the dictionary.



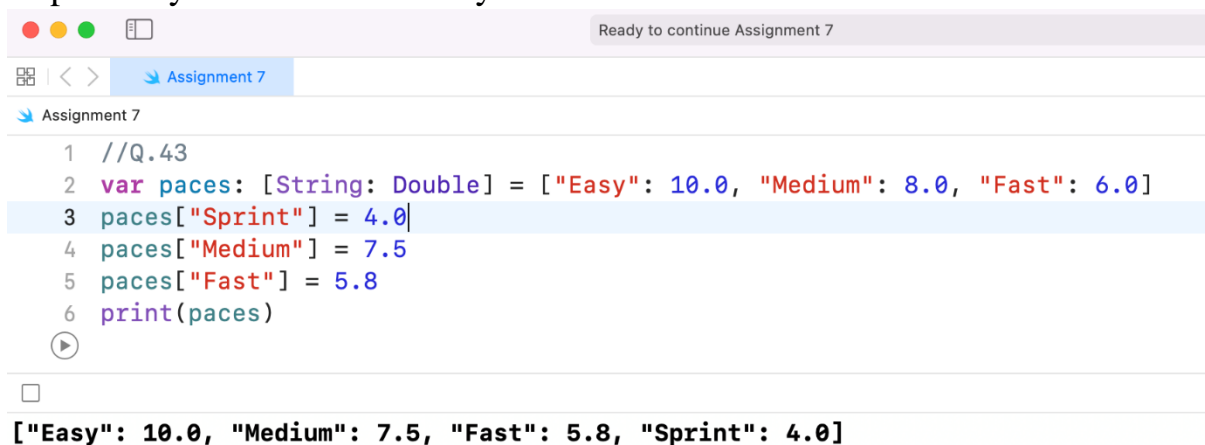
The screenshot shows a code editor window titled "Assignment 7" with a status bar indicating "Ready to continue Assignment 7". The code is as follows:

```
1 //Q.42
2 var paces: [String: Double] = ["Easy": 10.0, "Medium": 8.0, "Fast": 6.0]
3 paces["Sprint"] = 4.0
4 print(paces)
```

Below the code, the output of the program is displayed in a text area:

```
["Easy": 10.0, "Fast": 6.0, "Medium": 8.0, "Sprint": 4.0]
```

43. Imagine the user in question gets faster over time and decides to update his/her pacing on runs. Update the values of "Medium" and "Fast" to 7.5 and 5.8, respectively. Print the dictionary



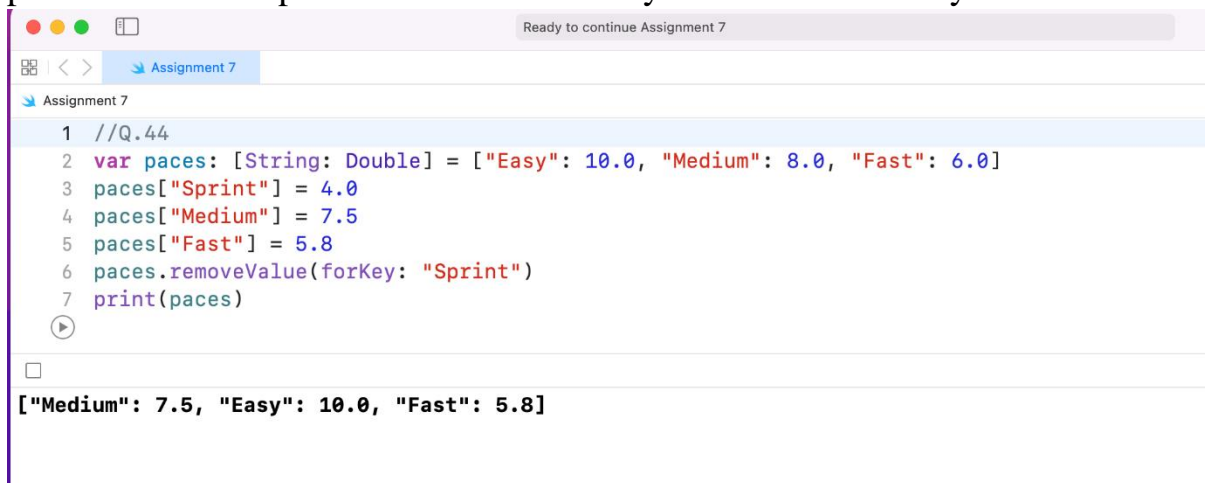
The screenshot shows a code editor window titled "Assignment 7" with a status bar indicating "Ready to continue Assignment 7". The code is as follows:

```
1 //Q.43
2 var paces: [String: Double] = ["Easy": 10.0, "Medium": 8.0, "Fast": 6.0]
3 paces["Sprint"] = 4.0
4 paces["Medium"] = 7.5
5 paces["Fast"] = 5.8
6 print(paces)
```

Below the code, the output of the program is displayed in a text area:

```
["Easy": 10.0, "Medium": 7.5, "Fast": 5.8, "Sprint": 4.0]
```

44. Imagine the user in question decides not to store "Sprint" as one his/her regular paces. Remove "Sprint" from the dictionary. Print the dictionary.



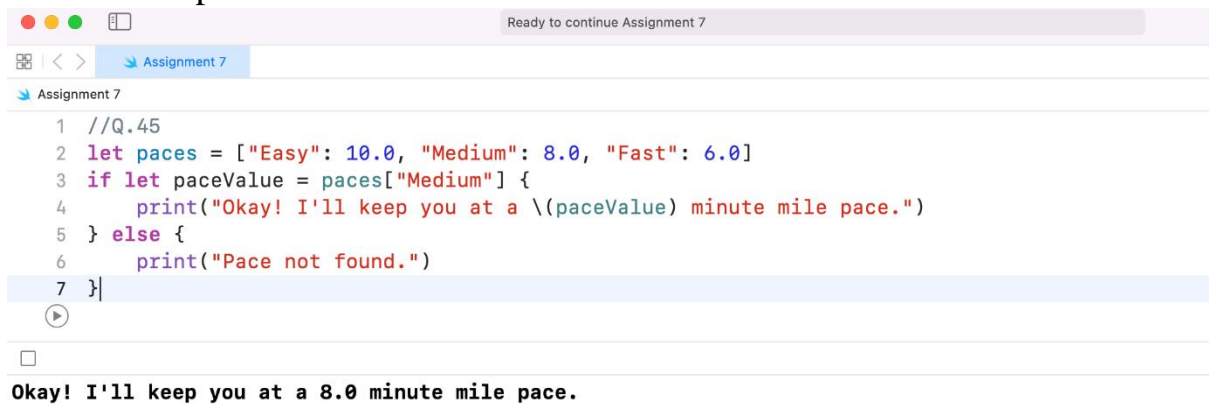
The screenshot shows a code editor window titled "Assignment 7" with a status bar indicating "Ready to continue Assignment 7". The code is as follows:

```
1 //Q.44
2 var paces: [String: Double] = ["Easy": 10.0, "Medium": 8.0, "Fast": 6.0]
3 paces["Sprint"] = 4.0
4 paces["Medium"] = 7.5
5 paces["Fast"] = 5.8
6 paces.removeValue(forKey: "Sprint")
7 print(paces)
```

Below the code, the output of the program is displayed in a text area:

```
["Medium": 7.5, "Easy": 10.0, "Fast": 5.8]
```

45. When a user chooses a pace, you want the app to print a statement stating that it will keep him/her on pace. Imagine a user chooses "Medium." Accessing the value from the dictionary, print a statement saying "Okay! I'll keep you at a minute mile pace."



```
1 //Q.45
2 let paces = ["Easy": 10.0, "Medium": 8.0, "Fast": 6.0]
3 if let paceValue = paces["Medium"] {
4     print("Okay! I'll keep you at a \(paceValue) minute mile pace.")
5 } else {
6     print("Pace not found.")
7 }
```

Okay! I'll keep you at a 8.0 minute mile pace.