**LOGIC 1**

**1) When squirrels get together for a party, they like to have cigars. A squirrel party is successful when the number of cigars is between 40 and 60, inclusive. Unless it is the weekend, in which case there is no upper bound on the number of cigars. Return True if the party with the given values is successful, or False otherwise.**

**cigar\_party(30, False) → False**

**cigar\_party(50, False) → True**

**cigar\_party(70, True) → True**

**def cigar\_party(cigars, is\_weekend):**

**if not is\_weekend:**

**if cigars>=40 and cigars<=60:**

**return True**

**else:**

**return False**

**else:**

**if cigars>=40:**

**return True**

**else:**

**return False**

**2) You are driving a little too fast, and a police officer stops you. Write code to compute the result, encoded as an int value: 0=no ticket, 1=small ticket, 2=big ticket. If speed is 60 or less, the result is 0. If speed is between 61 and 80 inclusive, the result is 1. If speed is 81 or more, the result is 2. Unless it is your birthday -- on that day, your speed can be 5 higher in all cases.**

**caught\_speeding(60, False) → 0**

**caught\_speeding(65, False) → 1**

**caught\_speeding(65, True) → 0**

**def caught\_speeding(speed, is\_birthday):**

**if is\_birthday:**

**if speed<=65:**

**return 0**

**elif speed>=66 and speed<=85:**

**return 1**

**elif speed>=86:**

**return 2**

**else:**

**if speed<=60:**

**return 0**

**elif speed>=61 and speed<=80:**

**return 1**

**elif speed>=81:**

**return 2**

**3) The number 6 is a truly great number. Given two int values, a and b, return True if either one is 6. Or if their sum or difference is 6. Note: the function abs(num) computes the absolute value of a number.**

**love6(6, 4) → True**

**love6(4, 5) → False**

**love6(1, 5) → True**

**def love6(a, b):**

**return (a==6 or b==6 or a+b==6 or abs(a-b)==6)**

**4)**

|  |
| --- |
| **You and your date are trying to get a table at a restaurant. The parameter "you" is the stylishness of your clothes, in the range 0..10, and "date" is the stylishness of your date's clothes. The result getting the table is encoded as an int value with 0=no, 1=maybe, 2=yes. If either of you is very stylish, 8 or more, then the result is 2 (yes). With the exception that if either of you has style of 2 or less, then the result is 0 (no). Otherwise the result is 1 (maybe).**  **date\_fashion(5, 10) → 2**  **date\_fashion(5, 2) → 0**  **date\_fashion(5, 5) → 1** |

**def date\_fashion(you, date):**

**if you<=2 or date<=2:**

**return 0**

**elif you>=8 or date>=8:**

**return 2**

**else:**

**return 1**

**5) Given 2 ints, a and b, return their sum. However, sums in the range 10..19 inclusive, are forbidden, so in that case just return 20.**

**sorta\_sum(3, 4) → 7**

**sorta\_sum(9, 4) → 20**

**sorta\_sum(10, 11) → 21**

**def sorta\_sum(a, b):**

**if a+b>=10 and a+b<=19:**

**return 20**

**else:**

**return a+b**

**6) Given a number n, return True if n is in the range 1..10, inclusive. Unless outside\_mode is True, in which case return True if the number is less or equal to 1, or greater or equal to 10.**

**in1to10(5, False) → True**

**in1to10(11, False) → False**

**in1to10(11, True) → True**

**def in1to10(n, outside\_mode):**

**if outside\_mode:**

**return (n<=1 or n>=10)**

**else:**

**return (n>=1 and n<=10)**

**7)**

|  |
| --- |
| **The squirrels in Palo Alto spend most of the day playing. In particular, they play if the temperature is between 60 and 90 (inclusive). Unless it is summer, then the upper limit is 100 instead of 90. Given an int temperature and a boolean is\_summer, return True if the squirrels play and False otherwise.**  **squirrel\_play(70, False) → True**  **squirrel\_play(95, False) → False**  **squirrel\_play(95, True) → True** |

**def squirrel\_play(temp, is\_summer):**

**if is\_summer:**

**return (temp>=60 and temp<=100)**

**else:**

**return (temp>=60 and temp<=90)**

**8)**

|  |
| --- |
| **Given a day of the week encoded as 0=Sun, 1=Mon, 2=Tue, ...6=Sat, and a boolean indicating if we are on vacation, return a string of the form "7:00" indicating when the alarm clock should ring. Weekdays, the alarm should be "7:00" and on the weekend it should be "10:00". Unless we are on vacation -- then on weekdays it should be "10:00" and weekends it should be "off".**  **alarm\_clock(1, False) → '7:00'**  **alarm\_clock(5, False) → '7:00'**  **alarm\_clock(0, False) → '10:00'** |

**def alarm\_clock(day, vacation):**

**if vacation:**

**if day==1 or day==2 or day==3 or day==4 or day==5:**

**return "10:00"**

**else:**

**return "off"**

**else:**

**if day==1 or day==2 or day==3 or day==4 or day==5:**

**return "7:00"**

**else:**

**return "10:00”**

**9)**

|  |
| --- |
| **Given a non-negative number "num", return True if num is within 2 of a multiple of 10. Note: (a % b) is the remainder of dividing a by b, so (7 % 5) is 2. See also: [Introduction to Mod](https://codingbat.com/doc/practice/mod-introduction.html)**  **near\_ten(12) → True**  **near\_ten(17) → False**  **near\_ten(19) → True** |

**def near\_ten(num):**

**res=num%10**

**return (res==1 or res==2 or res==8 or res==9 or res==0)**

**LOGIC 2**

|  |
| --- |
| **1) We want to make a row of bricks that is goal inches long. We have a number of small bricks (1 inch each) and big bricks (5 inches each). Return True if it is possible to make the goal by choosing from the given bricks. This is a little harder than it looks and can be done without any loops. See also: [Introduction to MakeBricks](https://codingbat.com/doc/practice/makebricks-introduction.html)**  **make\_bricks(3, 1, 8) → True**  **make\_bricks(3, 1, 9) → False**  **make\_bricks(3, 2, 10) → True**  **def make\_bricks(small, big, goal):**  **numberofinches=(small\*1)+(big\*5)**  **numberofsmallBricksNeeded=goal%5**  **return (goal<=numberofinches and numberofsmallBricksNeeded<=small)**  **2) Given 3 int values, a b c, return their sum. However, if any of the values is a teen -- in the range 13..19 inclusive -- then that value counts as 0, except 15 and 16 do not count as a teens. Write a separate helper "def fix\_teen(n):"that takes in an int value and returns that value fixed for the teen rule. In this way, you avoid repeating the teen code 3 times (i.e. "decomposition"). Define the helper below and at the same indent level as the main no\_teen\_sum().**  **no\_teen\_sum(1, 2, 3) → 6**  **no\_teen\_sum(2, 13, 1) → 3**  **no\_teen\_sum(2, 1, 14) → 3**  **def no\_teen\_sum(a, b, c):**  **return fif\_teen(a)+fif\_teen(b)+fif\_teen(c)**  **def fif\_teen(n):**  **if n>=13 and n<=19:**  **if n==15 or n==16:**  **return n**  **else:**  **return 0**  **else:**  **return n**  **3) We want make a package of goal kilos of chocolate. We have small bars (1 kilo each) and big bars (5 kilos each). Return the number of small bars to use, assuming we always use big bars before small bars. Return -1 if it can't be done.**  **make\_chocolate(4, 1, 9) → 4**  **make\_chocolate(4, 1, 10) → -1**  **make\_chocolate(4, 1, 7) → 2**  **def make\_chocolate(small, big, goal):**  **numberOfSmallBarsToUse=goal%5**  **numberOfBars=(small\*1)+(big\*5)**  **numberOfBigBarsGiven=big\*5**  **if numberOfBars<goal:**  **return -1**  **elif small<numberOfSmallBarsToUse:**  **return -1**  **elif numberOfSmallBarsToUse<numberOfBigBarsGiven:**  **if goal>numberOfBigBarsGiven:**  **return goal-numberOfBigBarsGiven**  **else:**  **return numberOfSmallBarsToUse**  **else:**  **return numberOfSmallBarsToUse** |
| **4) Given 3 int values, a b c, return their sum. However, if one of the values is the same as another of the values, it does not count towards the sum.**  **lone\_sum(1, 2, 3) → 6**  **lone\_sum(3, 2, 3) → 2**  **lone\_sum(3, 3, 3) → 0**  **def lone\_sum(a, b, c):**  **if a==b and b==c and c==a:**  **return 0**  **elif a==b:**  **return c**  **elif a==c:**  **return b**  **elif b==c:**  **return a**  **else:**  **return a+b+c**  **5)** |
| **For this problem, we'll round an int value up to the next multiple of 10 if its rightmost digit is 5 or more, so 15 rounds up to 20. Alternately, round down to the previous multiple of 10 if its rightmost digit is less than 5, so 12 rounds down to 10. Given 3 ints, a b c, return the sum of their rounded values. To avoid code repetition, write a separate helper "def round10(num):" and call it 3 times. Write the helper entirely below and at the same indent level as round\_sum().**  **round\_sum(16, 17, 18) → 60**  **round\_sum(12, 13, 14) → 30**  **round\_sum(6, 4, 4) → 10**  **def round\_sum(a, b, c):**  **return (round10(a)+round10(b)+round10(c))**    **def round10(n):**  **rightMostNumber=n%10**  **if rightMostNumber<5:**  **roundofNumber=n-rightMostNumber**  **elif rightMostNumber>=5:**  **roundofNumber=n+(10-rightMostNumber)**  **return roundofNumber** |
| **6) Given 3 int values, a b c, return their sum. However, if one of the values is 13 then it does not count towards the sum and values to its right do not count. So for example, if b is 13, then both b and c do not count.**  **lucky\_sum(1, 2, 3) → 6**  **lucky\_sum(1, 2, 13) → 3**  **lucky\_sum(1, 13, 3) → 1**  **def lucky\_sum(a, b, c):**  **if not a==13 and not b==13 and not c==13:**  **return a+b+c**  **elif a==13:**  **return 0**  **elif b==13:**  **return a**  **elif c==13:**  **return a+b** |
| **7) Given three ints, a b c, return True if one of b or c is "close" (differing from a by at most 1), while the other is "far", differing from both other values by 2 or more. Note: abs(num) computes the absolute value of a number.**  **close\_far(1, 2, 10) → True**  **close\_far(1, 2, 3) → False**  **close\_far(4, 1, 3) → True**  **def close\_far(a, b, c):**  **a\_b\_diff = abs(a - b)**  **a\_c\_diff = abs(a - c)**  **b\_c\_diff = abs(b - c)**  **return ((a\_b\_diff <= 1 and a\_c\_diff >= 2 and b\_c\_diff >= 2) !=(a\_c\_diff <= 1 and a\_b\_diff >= 2 and b\_c\_diff >= 2))** |
|  |