Iteration II

Meta Activity: Team Disruptions

Common disruptions to learning in teams include: talking about topics that are off-task, teammates answering questions on their own, entire teams working alone, limited or no communication between teammates, arguing or being disrespectful, rushing to complete the activity, not being an active teammate, not coming to a consensus about an answer, writing incomplete answers or explanations, ignoring ideas from one or more teammates.

1. Pick four of the disruptions listed above. For each one, find something from the role cards that could help improve the team's success. Use a different role for each disruption.

a) Manager:

limited communication between teammates

b) Presenter:

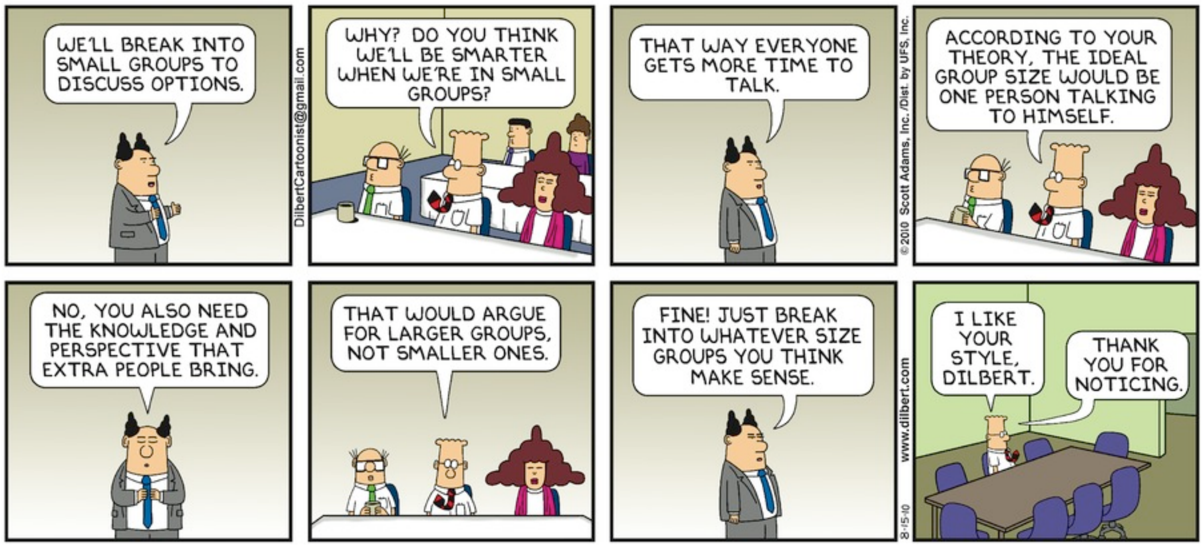
ignoring ideas from one or more teammates

c) Recorder:

writing incomplete answers or explanations

d) Reflector:

teammates answering questions on their own


WE'LL BREAK INTO SMALL GROUPS TO DISCUSS OPTIONS.
WHY? DO YOU THINK WE'LL BE SMARTER WHEN WE'RE IN SMALL GROUPS?
THAT WAY EVERYONE GETS MORE TIME TO TALK.
ACCORDING TO YOUR THEORY, THE IDEAL GROUP SIZE WOULD BE ONE PERSON TALKING TO HIMSELF.
NO, YOU ALSO NEED THE KNOWLEDGE AND PERSPECTIVE THAT EXTRA PEOPLE BRING.
THAT WOULD ARGUE FOR LARGER GROUPS, NOT SMALLER ONES.
FINE! JUST BREAK INTO WHATEVER SIZE GROUPS YOU THINK MAKE SENSE.
I LIKE YOUR STYLE, DILBERT.
THANK YOU FOR NOTICING.
DilbertCartoonist@gmail.com
©2010 Scott Adams, Inc./Dist. by UFS, Inc.
www.dilbert.com
8-15-10

Dilbert by Scott Adams. © Andrews McMeel Syndication. http://dilbert.com/strip/2010-08-15

Functions and for loops practice

2. Evaluate the following for loop:

```
def main() -> None:
    for i = 0 to 5:
        print(i, end = "")
main()
```

syntax error

3. What is the output of the following code snippet? Draw the loop table for each for loop.

```
def main() -> None:
    for i in range(4):
        print("i:", i+1)

    for j in range(4,0,-1):
        print("j: "+str(j))
main()
```

i	Output
0	i: 1
1	i: 2
2	i: 3
3	i: 4

j	Output
4	j: 4
3	j: 3
2	j: 2
1	j: 1

4. What is the output for `print_pattern('cat',3)`?

```
def print_pattern(word: str, copies: int) -> None:
    phrase = " * "
    for count in range(copies):
        phrase = phrase + word + " * "
        print(phrase)
```

count	phrase (after line 4)
0	'* cat *'
1	'* cat * cat *'
2	'* cat * cat * cat *'

5. What is the output for `print_pattern(3,'hi')`?

```python
def print_pattern(cnt: int, instr: str) -> None:
    mystr = "Be"
    for i in range(cnt):
        mystr = mystr + instr * i + "!" * (cnt-i)
        print(mystr)
```

i	mystr (after line 4)
0	'Be!!!'
1	'Be!!!hi!!'
2	'Be!!!hi!!hihi!'

6. What is the output for `print_pattern(3)`?

```python
def ants(repeat: int) -> None:
    ants = 1
    for i in range(repeat):
        print('The ants go marching', i, 'by', ants)
        ants = ants * 2
        print('hurrah ' * (i + 1))
    print('And ' + str(ants) + ' ants go marching down into the ground')
```

i	Output (line 4)	ants (after line 5)	Output (line 6)
0	'The ants go marching 0 by 1'	2	hurrah
1	'The ants go marching 1 by 2'	4	hurrah hurrah
2	'The ants go marching 2 by 4'	8	hurrah hurrah hurrah

```
The ants go marching 0 by 1
hurrah
The ants go marching 1 by 2
hurrah hurrah
The ants go marching 2 by 4
hurrah hurrah hurrah
And 8 ants go marching down into the ground
```

7. Write a function called `cheer` that asks a user for his/her name and prints a “cheer” in a particular format. If the user types ”Fred”, the output should be:

```
F !
rr !
eee !
dddd !
```

That is, the program should print a number of copies of each letter of the name on a separate line. The number of copies should increase with the number of lines printed. Each line should end with an exclamation point – and there should be no space between the exclamation point and the letters of the name. **Hint**: nested loops are not needed for this problem. As another example, if the user types the name ”Abi”, the output should be:

```
A !
bb !
i i i !
```

```python
def cheer(name:str) -> None:
    count = 1
    for character in name:
        print(count * character + "!")
        count += 1
```

8. Write a function called `cheer` that asks a user for a team’s name and prints a “cheer” in a particular format. That is, the program should print, on separate lines, each letter of the name with `"Gimme an"` before it, and an exclamation point afterward, then print one line with the team’s name repeated for the same number of letters in the name, with exclamation points after each occurrence of the name. If the user types ”Gate”, the output should be:

```
Gimme an G !
Gimme an a !
Gimme an t !
Gimme an e !
Gate!Gate!Gate!Gate!
```

```python
def cheer(name:str) -> None:
    repeats = ''
    for character in name:
        print ("Gimme an " + character + "!")
        repeats = repeats + name + '!'
    print(repeats)
```

9. Write a function called get_int that takes in a list of digits, calculates and returns the number corresponding to the digits in the list.
For example, get_int([3,7,1]) must return 371.

```python
def get_int(digits: list) -> int:
    number=0
    for digit in digits:
      number = number * 10 + digit

    return number
```

10. Write a function that take in a list and a count and returns a list with count elements picked at random from the input list (with replacement):

```python
import random

def random_slice(l:list, c: int) -> list:
    rl = []
    for i in range(c):
        rl.append(random.choice(l))
    return rl
```

11. Write a function that take in a list and a count and returns a list with count elements picked at random from the input list (without replacement):

```python
import random

def random_slice(l:list, c: int) -> list:
    rl = []
    for i in range(c):
        element = random.choice(l)
        rl.append(element)
        l.remove(element)
    return rl
```

12. Write a function `doubleReverse` that takes a list as a parameter. You can assume that every item in the list is a string. The function should return a new list that contains the strings in reverse order and with each string reversed. You can assume the list is non-empty.
Example: `doubleReverse(['moo', 'cow', 'moo!'])` returns `['!oom', 'woc', 'oom']`.

```python
def reverse(s):
    new_s = ""
    for i in range(1, len(s)+1):
        new_s += s[-i]
    return new_s

def doubleReverse(lst):
    """
    Expects a lst of strings and returns a new list that
    contains the strings in reverse order and with each
    string reversed.
    >>> doubleReverse(['moo', 'cow', 'moo!'])
    ['!oom', 'woc', 'oom']
    """
    new_lst = []
    for i in range(1, len(lst)+1):
        new_lst += [reverse(lst[-i])]
    return new_lst
```