2

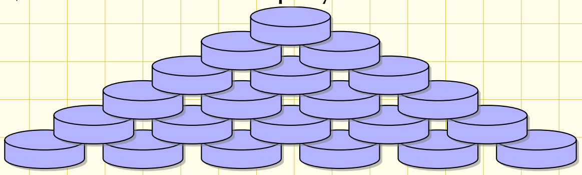
Take Away

Introduction

In this worksheet we will create a version of the Take Away game:

Take Away Game

Game consists of a pile of coins with two players taking turns to play. At each turn the player decides to take 1, 2 or 3 coins. The player who take the last coin wins.



We will build a number of versions of this game — each version gets better and more complicated! You can choose what type of improvements/extensions you find more interesting. I don't expect that we will get all versions completed.

① **Basic game**

We start by implementing the basic game where the computer is pretty stupid and only takes one coin each time.

② **Stop humans cheating!**

A big issue when writing programs that interact with people is that we cheat! Or to put it more nicely, we humans, don't always follow instructions as well as we should. So we need to write code to check that the human player's input is valid.

③ **If can't get smarter then go random**

Currently our computer player is pretty stupid (it always takes one coin). There is no challenge here. We want it to be better than that. In the next version we will make it better, but in this version we will — using just one line of code — make it **appear** much smarter by adding randomness.

④ **Get smarter**

In this version we will make the computer smarter — this is often called artificial intelligence — by enabling it to make decisions on how many coins to take based on the number of coins remaining in the game.

At this point we could go in many different directions:

- We could look at variations of the game. For example, we could change the rules regarding how many coins we are allowed to remove. To make this really hard we could have gaps, such as you are allowed to take 1, 3, or 4 coins but not allowed to take 2.
- Or we could go graphical — this will be the focus of next handout.

CoderDojo, Tramore, Waterford. (`kmurphy@wit.ie`)

1 Basic game

We start by implementing the basic game where the computer is pretty stupid and only takes one coin each time. The main items to note here are:

☞ A game will have three stages:

1. Setup game
2. Play game
3. Report result at end of game.

So to start, open a new file using menu **File→New** and enter the following comments

```
# initialise game

# play the game

# output result of game
```

Save this script as `TakeAway_Basic.py`. Right now the script does nothing! Comments are only there to help us.

☞ In the setup (initialisation) stage we will need create a variable to store the number of coins left in the game. And to make the game more interesting/difficult we want to start with a random number of coins, say between 15 and 25 coins. So we import the python module `random`. This module contains functions for generating random numbers.

```
import random
```

Then we generate a random integer[1]and store it in a variable, `coins`, for later using

```
coins = random.randint(15, 25)
```

Next, we need some way of recording whose is next to move. To use this we use another variable, `isComputerMove`, as follows

```
isComputerMove = False
```

The identifier `isComputerMove` will store `True` or `False` depending on which player is next to move.

OK, to finish off the initialisation stage we need to output the game instructions. I have not done that, but only have a `print` command to remind me that this needs to be done.

✔ Lets check our code ... at this point, the initialisation/setup stage looks like

```
# initialise game
import random
coins = random.randint(15, 25)
isComputerMove = False

print ("Instructions go here ...\n\n")
```

[1]An **integer** is a whole number.

☞ Use a **while** loop to keep playing until game ends.

☞ Last week we used a **for** loop to repeat a set of commands, today we will use a **while** loop. A **while** loop make mores sense here because we want to keep repeating a set of commands until the game ends and not just a fixed number of times. We start with

```
# play the game

while coins>0:

    print("The pile contains %s coins." % coins)
```

If you run the above code, it will print the message forever … well until end of class or you stop it.

☞ Use a **if** statement to determine which player should move.

```
    if isComputerMove:
        move = 1
        print ("I'm taking %s coins" % move)
    else:
        move = int(input("How many coins do you want? "))
        print ("You're taking %s coins" % move)
```

The above code needs some explanation:

- We are using the variable `move` to store the number of coins to take in the current move.
- In the computer's turn, we are simply setting `move` to 1, so the computer will always take one coin and will be very easy to beat. We will fix this later.
- In the human's turn, we get the user's move using **input** command This gives the move back as a string so we use **int** command to convert this to an integer.
- Notice that we print out the user's move — this is good practice, as a user may have pressed the wrong key.
- We use boolean variable, `isComputerMove`, to remember whose turn it is — human or computer.

☞ Now that we have the computer or human move, we must use it to update the amount of coins remaining

```
    coins = coins - move
```

☞ Finally for the play game stage we need to add code that switches players every turn

```
    isComputerMove = not isComputerMove
```

✔ Lets check our code ... at this point, the play game stage looks like

```python
# play the game

while coins>0:

    print("The pile contains %s coins." % coins)

    if isComputerMove:
        move = 1
        print ("I'm taking %s coins" % move)
    else:
        move = int(input("How many coins do you want? "))
        print ("You're taking %s coins" % move)

    coins = coins - move

    isComputerMove = not isComputerMove
```

☞ The third stage, output result of game is easy ... the rules state that the last player to play wins, so the next player to play loses. We can use `isComputerMove` to decide which message to output.

```python
# output result of game

if isComputerMove:
    print ("\nYou moved last. you win")
else:
    print ("\nI moved last. you lose")
```

At this point we have a working game. It is easy to win and it is even easier to cheat, but hey, it is still a game. Next we will see about improvements.

We are Done !

2 Stop humans cheating!

The main idea here is that we should repeatedly ask for input, then check the input that the human player entered and keep asking and asking for input until we get valid input. Rather than giving you code I have written the structure in a mixture of python and English and you have to translate it into python.

```
# human move -- need to check input because humans cheat !
while True:
    # TODO write code get move
    if : # TODO write code check if human tried to take too many coins
        # TODO write message scolding player
    elif move not in [1,2,3]:
        # TODO write message scolding player
    else:
        break
```

3 If can't get smarter then go random

OK, so we can no longer cheat ... but the game is still too easy to win. The computer always take one coin. We want to add some intelligence. Depending on the game, adding intelligence can be a difficult task. However here we can "cheat" too, and we can fake intelligence by inserting some randomness to the computer's move.
If you look back to the start of our code where we setup the game using a random number of coins, you will see that generating a random move is easy. In fast, it is so easy I won't insult you by giving the line of code that you need, but only say that you need to replace the `move = 1` line in the code for the computer's move:

```
    if isComputerMove:
        move = 1
        print ("I'm taking %s coins" % move)
```

by a call to `random.randint`. Remember that `random.randint` generates a random integer between two given integers.
However there is one tiny, tiny, tiny problem — one that I forgot about entirely when wring this — you need to make sure that the computer does not try to take move coins that the amount of coins remaining. I used the **`min`** function for this.

4 A bit of housekeeping (programming-wise)

Notice that our game loop is getting longer and more complicated. Before we add any more "improvements" we should restructure our code so that we can deal with the computer move and the human move separately from the game loop. We will use functions for this.
At end of setup/initialisation stage we define function that store the code for the computer's move

```python
def getComputerMove(coins):
    move = random.randint(1, min(3,coins))
    return move
```

and a second function for the human's move

```python
def getHumanMove(coins):
    # human move -- need to check input because humans cheat !
    while True:
        move = int(input("How many coins do you want to take (1, 2 or 3)? "
        if move>coins:
            print ("The pile only contains %s coins. Try again" % coins)
        elif move not in [1,2,3]:
            print ("Expected a move of '1', '2' or '3'.")
        else:
            break
    return move
```

We can then simplify the game loop as follows

```python
    if isComputerMove:
        move = getComputerMove(coins)
        print ("I'm taking %s coins" % move)
    else:
        move = getHumanMove(coins)
        print ("You're taking %s coins" % move)
```

5 Get smarter

OK, so now with the code reorganised, let's make the computer smarter ...
This step is a bit harder. You will need play the game a few times and develop a strategy. We will then help you code that strategy, but developing the strategy is your problem!