Steam API Tutorial

Denise Cilia
X81000791
Social Media Management
A.A 21/22

Introduzione

Steam è una piattaforma sviluppata da Valve Corporation che si occupa di distribuzione digitale, di gestione dei diritti digitali, di modalità di gioco multigiocatore e di comunicazione. Viene usata per gestire e distribuire una vasta gamma di giochi (alcuni esclusivi) e il loro relativo supporto.

La piattaforma mette a disposizione un'API chiamata Steamworks, che consente agli sviluppatori di usufruire di funzionalità offerte da Steam, tra cui trofei di gioco, microtransazioni e un supporto per i contenuti creati da un utente tramite lo Steam Workshop. Oltre a ciò la piattaforma fornisce anche una piccola selezione di software per la progettazione, anime e film.

API

L'API è di tipo REST e contiene metodi pubblici a cui è possibile accedere da qualsiasi applicazione in grado di effettuare una richiesta HTTP a `api.steampowered.com`, come client oppure server di gioco.

Le chiamate dell'API hanno la seguente forma `api.steampowered.com` / `interfaceName` / `methodName` / `vVersion` / `?key=apiKey`. Il risultato delle richieste potrà essere uno dei tre seguenti formati: JSON, XML, VDF (Valve Data Format). Se nessun formato viene specificato nella richiesta, l'API darà di default un risultato di tipo JSON.

La lista dei metodi ed interfacce pubblicamente fornite è la seguente:

ISteamNews: Steam offre metodi per ottenere news e aggiornamenti per ciascun gioco sulla piattaforma Steam.

ISteamUserStats: Steam offre metodi per ottenere le statistiche globali ed informazioni di un gioco in particolare.

ISteamUser: Steam offre delle chiamate API per ottenere informazioni generiche e specifiche degli utenti Steam.

In generale troviamo la documentazione completa e ufficiale fornita da Steam a questo link: Docs

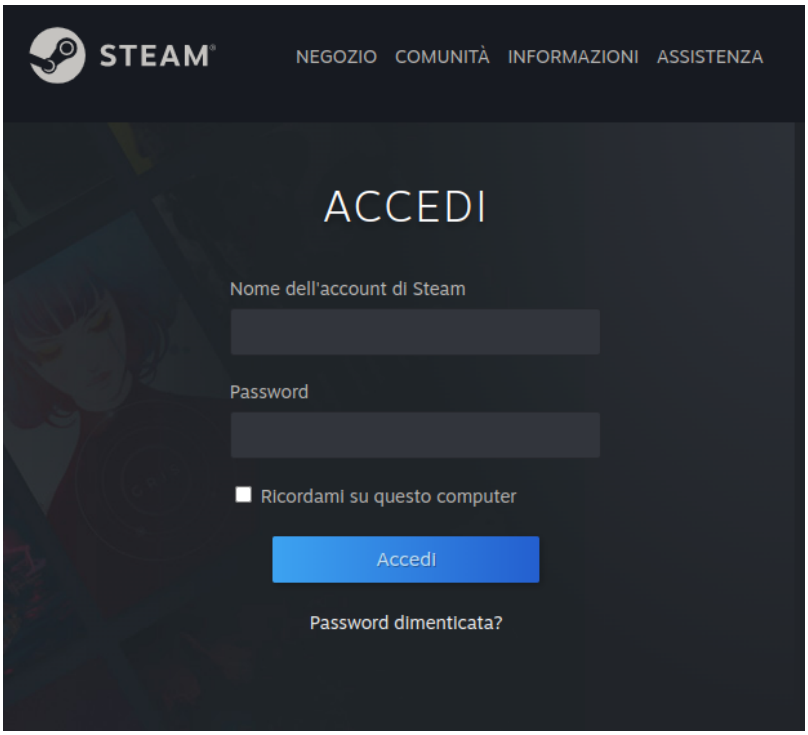
---

Accesso all'API

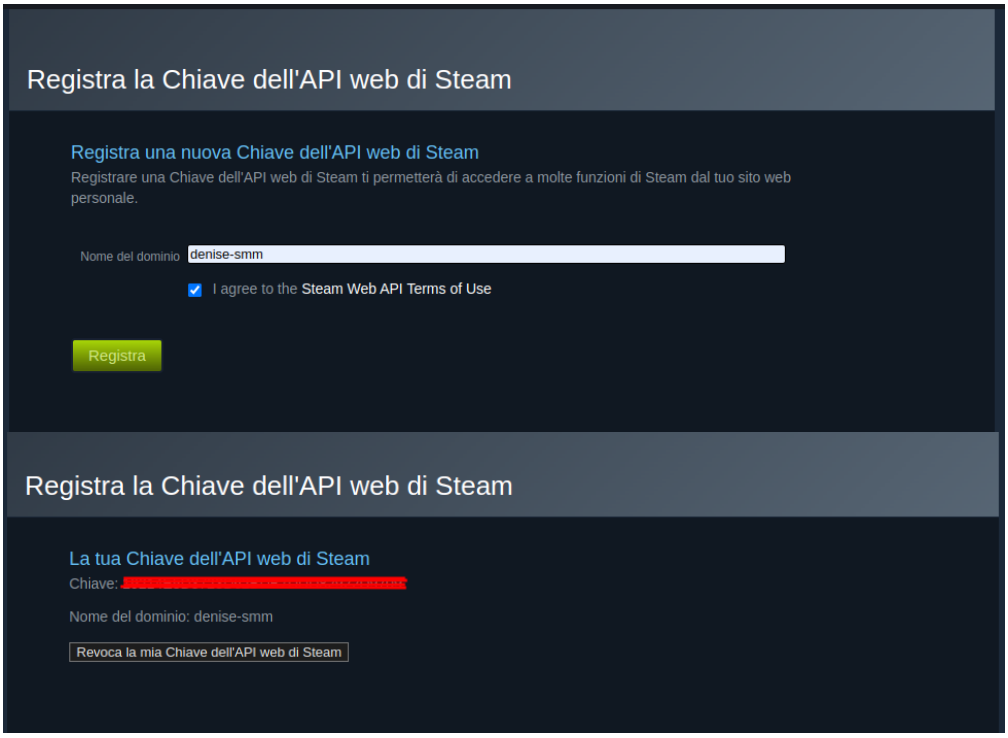
Per poter utilizzare le API dobbiamo innanzitutto avere un profilo utente Steam. Qualora non si possedesse un profilo utente è possibile crearlo in modo rapido e intuitivo cliccando qui.

Effettuiamo dunque l'accesso a Steam per ottenere le chiavi per poter utilizzare l'API.

La schermata che visualizziamo è la seguente:



Una volta inserite le nostre credenziali avremo la possibilità di ottenere una chiave inserendo un nome di dominio (può essere qualsiasi è una scelta puramente formale).



È bene notare che Valve segue una politica molto ristretta per quanto riguarda l'autenticazione tramite OAuth (vedi sezione Introduzione). Per poter ottenere il `clientId` utile all'implementazione della richiesta di autorizzazione bisogna contattare Valve e fornire le informazioni richieste.

Questo studio è fortemente concentrato su una serie di dati ad accesso **pubblico**. È dunque fondamentale che il profilo preso in studio abbia una privacy settata su **pubblico**. Ottenuta la nostra chiave pubblica possiamo adesso mostrare una sezione di codice utile ad ottenere diversi dati utili ai fini di uno studio di profilazione.

Inseriamo in questa variabile la nostra chiave da sviluppatore:

In [1]:
```
apiKey = 'YOUR_API_KEY'
```

Steam API Wrapper

Prima di procedere alla spiegazione sull'uso dell'API Steam mostriamo un Wrapper di Steam scritto in Python che permette di nascondere le richieste HTTP. Si tratta di una libreria che rende semplice l'accesso ad alcune informazioni dell'utente; possiamo accedere velocemente alla lista di amici dell'utente, alla lista di giochi generica e alla lista dei titoli giocati recentemente e altre funzionalità.

Vediamo qualche piccolo esempio:

In [2]:
```
import steamapi

steamapi.core.APIConnection(api_key=apiKey, validate_key=True)

steam_user = steamapi.user.SteamUser(userurl = "YOUR_NICKNAME")

steam_user.friends
```

Out[2]:
```
[<SteamUser "Mip" (76561198044574276)>,
 <SteamUser "Louphelle" (76561198051550543)>,
 <SteamUser "Jack King Goff" (76561198057136350)>,
 <SteamUser "TakaEyes" (76561198072068013)>,
 <SteamUser "Salvo4682" (76561198089183727)>,
 <SteamUser "Pancake alla Carciofina" (76561198104736604)>,
 <SteamUser "MarkTenma" (76561198106411612)>,
 <SteamUser "Swaccney" (76561198119254643)>,
 <SteamUser "Kepler" (76561198127997667)>,
 <SteamUser "CWS_97" (76561198142803553)>,
 <SteamUser "Riky68" (76561198155403000)>,
 <SteamUser "Kuroshiroix" (76561198188057123)>,
 <SteamUser "V" (76561198241697476)>,
 <SteamUser "CristaDoloris" (76561198269433080)>,
 <SteamUser "isidorodibartolo97" (76561198274365212)>,
 <SteamUser "ninni.cicci96" (76561198312852812)>,
 <SteamUser "mononoke2" (76561198346442638)>,
 <SteamUser "LaPHToxiNe.gamehag.com" (76561198349223210)>,
 <SteamUser "Niser" (76561198377913347)>,
 <SteamUser "gabry05" (76561198410768083)>,
 <SteamUser "R3DFox21" (76561198430290738)>,
 <SteamUser "Arisaka117" (76561198442382861)>,
 <SteamUser "Lince" (76561198821148601)>,
 <SteamUser "trepuntini" (76561198821700520)>,
 <SteamUser "antho99" (76561198839493532)>,
 <SteamUser "14giovy" (76561198844700805)>,
 <SteamUser "Davidpool" (76561198867378314)>,
 <SteamUser "centaurus97" (76561199006490525)>,
 <SteamUser "Schillinger19" (76561199207632683)>]
```

Tra parentesi troviamo indicato l'**id64** univoco per ciascun utente.

In [3]:
```
steam_user.recently_played
```

Out[3]:
```
[<SteamApp "Resident Evil 7 Biohazard" (418370)>,
 <SteamApp "Don't Starve Together" (322330)>,
 <SteamApp "Thronebreaker: The Witcher Tales" (973760)>,
 <SteamApp "BLEACH Brave Souls - 3D Action" (1201240)>,
 <SteamApp "Resident Evil 2" (883710)>,
 <SteamApp "Yu-Gi-Oh!  Master Duel" (1449850)>,
 <SteamApp "Dead Island Definitive Edition" (383150)>,
 <SteamApp "Orcs Must Die! 2" (201790)>,
 <SteamApp "Goat Simulator" (265930)>,
 <SteamApp "Undertale" (391540)>,
 <SteamApp "The Binding of Isaac: Rebirth" (250900)>,
 <SteamApp "Skul: The Hero Slayer" (1147560)>,
 <SteamApp "Five Nights at Freddy's" (319510)>,
 <SteamApp "Borderlands GOTY Enhanced" (729040)>,
 <SteamApp "The Binding of Isaac" (113200)>,
 <SteamApp "Home Sweet Home" (617160)>,
 <SteamApp "Wallpaper Engine" (431960)>]
```

Allo stesso modo, troviamo tra parentesi l'**id** univoco di ciascun gioco.

Purtroppo la libreria non è ben documentata, disponiamo di una piccolissima wiki incentrata principalmente sui metodi che riguardano l'utente. \ Per questo studio si decide dunque di procedere utilizzando in via diretta l'API ufficiale fornita da Steam.

Utilizzo dell'API

Definiamo una funzione `sendRequest`.

Ritorna una risposta in formato json di una generica richiesta GET utilizzando parametri da definire. Quest'ultimi saranno `url` che è una stringa e `parameters` che rappresenta un dizionario utilizzato per memorizzare i valori in coppie del tipo key:valore. In questo caso `parameters` sarà utilizzata per impostare i vari parametri per personalizzare la chiamata API su un determinato url.

In [4]:
```
import requests

def sendRequest(url, parameters = None):
    requests.adapters.DEFAULT_RETRIES = 5
    result = requests.get(url = url, params = parameters, timeout = 5)

    return result.json()
```

Utilizziamo dunque `sendRequest` per poter ottenere l'`id64` dell'utente Steam. Con questo risultato saremo in grado di risolvere diverse query che riguardano l'utente in particolare.

In [5]:
```
parameters = {'key':apiKey, 'vanityurl':'YOUR_NICKNAME'}
url = "http://api.steampowered.com/ISteamUser/ResolveVanityURL/v0001/"

result = sendRequest(url, parameters)
personalUID = result['response']['steamid']

personalUID
```

Out[5]:
```
'76561198093585873'
```

Definiamo adesso una funzione `userSummaries` in grado di dare in output i dati personali dell'utente, quali:

- nome
- nickname
- ultimo accesso
- città e paese di provenienza

In [6]:
```
from datetime import datetime

def userSummaries():
        url = "https://api.steampowered.com/ISteamUser/GetPlayerSummaries/v0
        params = {'key':apiKey, 'steamids':personalUID}
        result = sendRequest(url, params)
        result = result['response']['players']
        realName = result[0]['realname']
        nickname = result[0]['personaname']
        lastSeen = datetime.fromtimestamp(result[0]['lastlogoff'])
        country = str(result[0]['loccountrycode'])
        stateCode = str(result[0]['locstatecode'])
        url = "https://steamcommunity.com/actions/QueryLocations/" + country
        params = None

        location = sendRequest(url, params)
        for i in range(len(location)):
            if location[i]['cityid'] == result[0]['loccityid']:
```

```
                city = location[i]['cityname']

        print("Il nome dell'utente è " + realName + ", il suo nickname è " +
```

IPlayerService/GetOwnedGames

In [7]:

```
import pandas as pd
import math

def getAllGames():
    parameters = {'key':apiKey, 'include_appinfo':'TRUE', 'steamid':personal
    url = "https://api.steampowered.com/IPlayerService/GetOwnedGames/v0001/"
    list_of_games = sendRequest(url, parameters)
    list_of_games = list_of_games['response']['games']

    dictList = []
    recently_played = []

    for i in range(len(list_of_games)):
        try:
            list_of_games[i]['playtime_2weeks']
        except:
            recently_played.append("no")
        else:
            recently_played.append("yes")
        games = {
            'id':list_of_games[i]['appid'],
            'name':list_of_games[i]['name'],
            'hours_of_game':math.ceil((list_of_games[i]['playtime_forever']/
            'recently_played':recently_played[i]
        }
        dictList.append(games)

    df = pd.DataFrame(dictList)
    return df

data = getAllGames()
data
```

Out[7]:

	id	name	hours_of_game	recently_played
0	20	Team Fortress Classic	0	no
1	15710	Oddworld: Abe's Exoddus	0	no
2	9480	Saints Row 2	0	no
3	1250	Killing Floor	1	no
4	35420	Killing Floor Mod: Defence Alliance 2	0	no
...	...	...	...	...
291	46510	Syberia 2	0	no
292	204450	Call of Juarez Gunslinger	0	no
293	520720	Dear Esther: Landmark Edition	0	no
294	1577120	The Quarry	0	no
295	1515950	Capcom Arcade Stadium	0	no

296 rows × 4 columns

Definiamo una funzione `getAllGames` che ci permette di avere in output la lista di tutti i giochi in possesso, inserita all'interno di un dataframe. Le colonne sono id, name, hours_of_game e recently_played e descrivono rispettivamente:

- id univoco e identificativo per il gioco
- nome del gioco
- ore di gioco dell'utente
- se è stato giocato nelle ultime due settimane

Docs

AppDetails

In [8]:
```
from time import sleep

def getGenres():
    url = "https://store.steampowered.com/api/appdetails/"

    genre = []
    for i in range(len(data)):
        endstring = ''
        parameters = {'appids':data['id'][i]}
        result = sendRequest(url, parameters)
        tmp = data['id'][i]
        temp = f'{tmp}'
        try:
            result[temp]['data']['genres'][0]['description']
        except:
            genre.append('NA') # se non vi è alcuna specifica sul genere in
        else:
            for gen in range(len(result[temp]['data']['genres'])):
                endstring += result[temp]['data']['genres'][gen]['descriptio
                endstring += " " # potrebbe esserci più di un genere per gi
            genre.append(endstring)
        sleep(1) # utilizziamo una sleep per non inondare di richieste il s

    return genre

test = getGenres()
test
```

Out[8]:
```
['Action ',
 'Adventure ',
 'Action ',
 'Action ',
 'Action ',
 'Action Adventure ',
 'RPG ',
 'Action Adventure ',
 'Action Adventure ',
 'Action ',
 'NA',
 'Action ',
 'Adventure Casual Indie ',
 'Action Adventure Indie RPG ',
 'RPG ',
 'Action RPG ',
 'Action RPG ',
 'Action Adventure Indie RPG ',
 'Action ',
 'Action ',
 'Indie Strategy ',
 'Action ',
 'Action Adventure Indie RPG Strategy ',
 'Action Adventure Indie Simulation ',
 'Action RPG ',
 'Action ',
 'Action Adventure ',
 'Simulation Strategy ',
 'Adventure ',
 'Action Free to Play Massively Multiplayer ',
 'Action Free to Play Indie Massively Multiplayer RPG ',
 'Action Adventure Free to Play Massively Multiplayer RPG ',
 'Action Free to Play Massively Multiplayer ',
 'Action Free to Play Massively Multiplayer ',
 'Action Adventure Free to Play Indie Massively Multiplayer RPG ',
 'Action Adventure ',
 'Action Adventure Strategy ',
 'Action ',
 'NA',
 'Racing Sports ',
 'Strategy ',
 'Action Indie ',
 'Action ',
 'Action Massively Multiplayer Simulation ',
 'Action ',
 'Action Adventure ',
 'Action Indie RPG ',
 'Violent Action Adventure Indie ',
 'Action Adventure Indie ',
 'Adventure Indie ',
 'Action Adventure Indie ',
 'Action Adventure ',
 'Gore Indie Simulation ',
 'RPG ',
 'Action ',
 'Casual Indie ',
 'Indie RPG Strategy ',
 'RPG Simulation Strategy ',
 'Action Adventure Casual Indie RPG ',
 'Strategy ',
 'Action Adventure Casual Indie RPG ',
 'NA',
 'Adventure Casual Indie ',
 'Action Indie ',
```

```
 'Adventure Indie ',
 'Action ',
 'Action Adventure Indie ',
 'Action Adventure Indie ',
 'Action Casual Free to Play Indie ',
 'Action Adventure ',
 'Action Adventure ',
 'Action Adventure Indie ',
 'Action Adventure ',
 'Racing Sports ',
 'Action Adventure Free to Play Sports Strategy ',
 'Adventure Casual ',
 'Adventure Casual ',
 'Adventure Casual ',
 'Racing ',
 'Adventure Indie ',
 'Adventure ',
 'Indie ',
 'Action Indie RPG ',
 'Action Casual Indie ',
 'Indie RPG Simulation Early Access ',
 'Action Adventure Casual Free to Play Massively Multiplayer RPG ',
 'Action Adventure Casual Free to Play Indie ',
 'Indie Simulation ',
 'Action RPG ',
 'Racing Simulation ',
 'Action ',
 'Action ',
 'Action Adventure Indie ',
 'Action RPG ',
 'Adventure ',
 'Free to Play Indie Simulation Strategy ',
 'Action RPG ',
 'Action ',
 'Action Adventure ',
 'Action Free to Play Indie Massively Multiplayer ',
 'Action ',
 'Action Adventure ',
 'Action Adventure ',
 'Action Free to Play Indie RPG ',
 'Action Free to Play ',
 'Action Adventure RPG ',
 'Action Adventure Indie ',
 'Adventure Indie Simulation ',
 'Action ',
 'Action Adventure Indie RPG Strategy ',
 'Adventure Indie Simulation ',
 'Action ',
 'Action Adventure Indie ',
 'NA',
 'Action Free to Play Indie ',
 'Action Sports ',
 'Indie RPG ',
 'Action RPG ',
 'Action Free to Play Indie RPG Strategy ',
 'Indie RPG Simulation ',
 'Action Adventure Casual Free to Play Indie Massively Multiplayer RPG Simu
lation ',
 'Action Adventure ',
 'Action Adventure ',
 'Casual Indie Animation & Modeling Design & Illustration Photo Editing Uti
lities ',
 'Adventure Casual Indie RPG Simulation ',
 'Adventure Casual Free to Play Massively Multiplayer Simulation Sports Ear
```

```
ly Access ',
 'Action Free to Play ',
 'NA',
 'Action Adventure RPG ',
 'NA',
 'Adventure Indie Strategy ',
 'Adventure Casual Indie Simulation ',
 'Action ',
 'Action Indie Simulation Strategy ',
 'Action Adventure Indie Simulation ',
 'Action Adventure ',
 'Action ',
 'Action ',
 'Adventure Indie ',
 'Action Adventure ',
 'Action Indie Simulation ',
 'Action Adventure Indie ',
 'Racing Simulation Sports Strategy ',
 'Action Adventure Free to Play Indie RPG Strategy ',
 'RPG ',
 'Action Adventure Indie ',
 'Action Casual Free to Play Indie ',
 'Action Adventure Free to Play Massively Multiplayer RPG ',
 'Action Free to Play ',
 'Action Adventure Free to Play Massively Multiplayer RPG ',
 'Action Indie Sports ',
 'Indie Simulation Strategy ',
 'Action Adventure Casual Free to Play Massively Multiplayer RPG ',
 'Action Aventure ',
 'Action Indie ',
 'Free to Play Indie Simulation ',
 'Action Free to Play Indie ',
 'Action Adventure Massively Multiplayer RPG Simulation Strategy ',
 'Action Adventure Casual Indie ',
 'Action Indie ',
 'Free to Play Strategy ',
 'Action ',
 'Adventure Indie ',
 'Action Adventure Indie Strategy ',
 'Action Adventure Free to Play Indie Strategy ',
 'Casual Indie Simulation ',
 'Action Free to Play Massively Multiplayer Racing ',
 'Action Adventure Indie ',
 'Action RPG ',
 'Action Adventure Indie Simulation Strategy ',
 'Action Casual Free to Play Indie Simulation Sports Early Access ',
 'Casual Free to Play Massively Multiplayer RPG Sports Early Access ',
 'Action Adventure Indie ',
 'Action Free to Play Early Access ',
 'Action Adventure RPG ',
 'Casual Free to Play Indie ',
 'Action Adventure Indie RPG ',
 'Action Free to Play ',
 'NA',
 'Adventure Free to Play Indie ',
 'Action Free to Play Massively Multiplayer Strategy ',
 'Action Casual Indie ',
 'Indie Simulation Strategy ',
 'RPG ',
 'Action Free to Play Indie ',
 'Free to Play Indie Simulation ',
 'Casual Indie ',
 'NA',
 'Action Free to Play ',
```

```
'Action Free to Play ',
'Action Adventure Indie ',
'Simulation Strategy ',
'Action Free to Play Early Access ',
'NA',
'Action ',
'Action Adventure Free to Play Massively Multiplayer ',
'Massively Multiplayer ',
'Action Free to Play Indie ',
'Action ',
'Action Free to Play Early Access ',
'Adventure Free to Play ',
'Free to Play Indie RPG ',
'Action ',
'Strategy ',
'Action ',
'Action Free to Play Indie ',
'Action Free to Play Early Access ',
'Action Adventure Indie RPG Simulation ',
'Action Adventure RPG ',
'Racing Simulation Sports ',
'Action Indie Strategy ',
'Adventure Casual Free to Play Indie ',
'Action Free to Play ',
'Action Adventure Casual Indie Strategy ',
'Action ',
'Action Indie Simulation Strategy ',
'Adventure RPG ',
'Action ',
'Adventure ',
'Action ',
'Action Adventure Free to Play Indie RPG Strategy ',
'Action Indie Simulation Strategy ',
'Action Indie RPG Early Access ',
'Adventure Indie ',
'Ação ',
'Action Adventure Indie ',
'Indie ',
'Action Casual Free to Play Indie Strategy ',
'Strategy Early Access ',
'Action Free to Play Massively Multiplayer ',
'Action Indie ',
'Racing Simulation Sports ',
'Action Adventure Free to Play ',
'Free to Play RPG Simulation Strategy ',
'Action RPG ',
'Action Casual Free to Play RPG ',
'Action Indie RPG ',
'Action Adventure RPG Strategy ',
'NA',
'Action ',
'Action ',
'Action ',
'Adventure Indie ',
'Action Early Access ',
'Action Casual Indie Sports ',
'NA',
'Action Adventure ',
'Action Adventure ',
'Strategy ',
'Adventure Indie RPG ',
'Action Indie ',
'Indie Strategy ',
'Action Adventure Free to Play Indie RPG ',
```

```
 'Simulation ',
 'Adventure Free to Play Indie ',
 'Strategy ',
 'Action Casual Free to Play ',
 'Action Indie ',
 'Action ',
 'Action ',
 'NA',
 'NA',
 'NA',
 'Animation & Modeling Design & Illustration Education Software Training Ut
ilities Game Development ',
 'Action Adventure Indie ',
 'Action Adventure Indie ',
 'Action RPG Simulation ',
 'Free to Play Indie Early Access ',
 'Action Adventure Casual RPG ',
 'Free to Play Simulation Strategy ',
 'Adventure Free to Play Indie ',
 'Action RPG ',
 'Casual Free to Play Indie Massively Multiplayer RPG Simulation Early Acce
ss ',
 'Action Adventure Casual Free to Play Indie Massively Multiplayer ',
 'Adventure Free to Play Indie ',
 'Action Casual Free to Play Early Access ',
 'Action Adventure Indie Simulation Strategy ',
 'Strategy ',
 'Strategy ',
 'Adventure ',
 'Casual Simulation ',
 'Action ',
 'Strategy ',
 'Action Adventure Free to Play Indie Massively Multiplayer Simulation Stra
tegy ',
 'Adventure ',
 'Casual Simulation ',
 'Action Adventure Indie ',
 'Action Adventure Casual Indie Simulation Strategy ',
 'Action Adventure Free to Play Massively Multiplayer ',
 'Strategy ',
 'Adventure Casual ',
 'Adventure ',
 'Action ',
 'Adventure Casual Indie ',
 'Adventure ',
 'Action Free to Play ']
```

La funzione `getGenres` ci permette di avere in output una lista di generi associati ai giochi posseduti dall'utente. Necessitiamo dell'id univoco di ciascun gioco per poter ottenere l'informazione che ci interessa. Questa verrà in seguito utilizzata per calcolare un grafico a torta in grado di mostrarci i generi preferiti, più giocati dall'utente.

In [18]:
```
data['genres'] = test

def mostPlayed():
    result = data.loc[(data['recently_played'] == "yes") & (data['hours_of_
    return result

most_played = mostPlayed()
most_played
```

Out[18]:

	id	name	hours_of_game	recently_played	genres
17	113200	The Binding of Isaac	99	yes	Action Adventure Indie RPG
97	250900	The Binding of Isaac: Rebirth	392	yes	Action
110	322330	Don't Starve Together	942	yes	Adventure Indie Simulation
219	883710	Resident Evil 2	62	yes	Action
230	1147560	Skul: The Hero Slayer	50	yes	Action Indie
235	1201240	BLEACH Brave Souls - 3D Action	634	yes	Action Casual Free to Play RPG
269	1449850	Yu-Gi-Oh! Master Duel	113	yes	Free to Play Simulation Strategy

Estraiamo con la funzione `mostPlayed` i titoli giocati nelle ultime due settimane e le cui ore di gioco totali sono maggiori o uguali a 50. Otterremo così una lista di giochi dalla quale sarà possibile estrarre gli achievements del giocatore.

ISteamUserStats/GetGlobalAchievementPercentagesForApp

In [19]:
```
def globalPercentageOfAchievement(n):
    globalPercentage = []
    url = 'https://api.steampowered.com/ISteamUserStats/GetGlobalAchievement
    params = {'gameid':most_played.iloc[n]['id'], 'format':'json'}
    result = sendRequest(url, params)
    result = result['achievementpercentages']['achievements']
    for i in range(len(result)):
        globalPercentage.append(result[i]['percent'])

    return globalPercentage
```

Docs.

ISteamUserStats/GetPlayerAchievements

In [20]:
```
from datetime import datetime

def getAchievements(n):
    url = 'http://api.steampowered.com/ISteamUserStats/GetPlayerAchievements
    params = {'appid':most_played.iloc[n]['id'], 'key':apiKey, 'steamid':pe
    result = sendRequest(url, params)
    result = result['playerstats']['achievements']
    percentage = globalPercentageOfAchievement(n)
    dictList = []
    for i in range(len(result)):
        for j in range(len(result[i])):
            tmp = {
                'name':result[i]['apiname'],
                'achieved':result[i]['achieved'],
                'timestamp':datetime.fromtimestamp(result[i]['unlocktime'])
                'global_percentage_of_unlock':math.ceil(percentage[i])/100
            }
        dictList.append(tmp)
    result = pd.DataFrame(dictList)
```

```
    return result
```

Docs.

Supponendo di conoscere la posizione di un titolo all'interno della libreria di giochi, indichiamo con *N* tale posizione e la utilizziamo come parametro nelle funzioni `globalPercentageOfAchievement` e `getAchievements`.

- **globalPercentageOfAchievement** ci permette di ottenere una panoramica globale sugli achievements di un gioco specifico in percentuali. In altre parole è la percentuale generale di sblocco dei trofei studiata sulla base delle statistiche di sblocco di tutti i giocatori nel mondo per quel particolare gioco.
- **getAchievements** ritorna la lista di achievements dell'utente, ottenuti e non, utilizzando l'id univoco ed identificativo del gioco in particolare.

Uniamo il risultato della prima e della seconda funzione in un unico dataframe che presenta le seguenti colonne: **name** (nome dell'achievement), **achieved** (valore booleano che indica se l'achievement *i* è stato sbloccato), **timestamp** (indica la data e l'ora in cui è stato sbloccato l'achievement) e **global_percentage_of_unlock** (mostra la percentuale generale di sblocco per l'achievement *i*

Vediamo degli esempi:

In [21]:
```
n = 3 # Resident Evil 2
achievements = []
gameName = []
gameName.append(most_played.iloc[n]['name'])
achievements.append(getAchievements(n))
achievements[0].head()
```

Out[21]:

	name	achieved	timestamp	global_percentage_of_unlock
0	NEW_ACHIEVEMENT_1_1	1	2020-11-27 19:37:13	0.95
1	NEW_ACHIEVEMENT_1_2	1	2021-12-08 15:59:37	0.87
2	NEW_ACHIEVEMENT_1_3	1	2021-12-08 17:15:48	0.85
3	NEW_ACHIEVEMENT_1_4	1	2021-12-08 17:35:48	0.83
4	NEW_ACHIEVEMENT_1_5	1	2021-12-09 15:41:55	0.83

In [22]:
```
n = 0 # The Binding of Isaac
gameName.append(most_played.iloc[n]['name'])
achievements.append(getAchievements(n))
achievements[1]
```

Out[22]:

	name	achieved	timestamp	global_percentage_of_unlock
0	The_Noose	1	2021-01-31 16:01:44	0.71
1	The_Nail	1	2021-02-01 13:59:11	0.63
2	The_Quarter	1	2021-02-01 21:30:55	0.60
3	The_Fetus	1	2021-02-02 18:45:06	0.57
4	Terrible	1	2021-02-03 13:01:47	0.49
...	...	...	...	...
94	psam	1	2021-08-27 00:55:45	0.02
95	peve	1	2021-08-23 12:50:03	0.02
96	pblue	1	2021-08-23 20:32:58	0.02
97	ppers	1	2021-08-27 00:55:45	0.02
98	egod	0	1970-01-01 01:00:00	0.02

99 rows × 4 columns

In [23]:
```
n = 6 # Yu-Gi-Oh! Master Duel
gameName.append(most_played.iloc[n]['name'])
achievements.append(getAchievements(n))
achievements[2]
```

Out[23]:

	name	achieved	timestamp	global_percentage_of_unlock
0	jp.konami.masterduel.ach.001	1	2022-01-19 15:36:42	0.89
1	jp.konami.masterduel.ach.002	1	2022-07-18 00:40:48	0.43
2	jp.konami.masterduel.ach.003	1	2022-02-18 14:20:50	0.40
3	jp.konami.masterduel.ach.004	1	2022-03-18 23:52:43	0.37
4	jp.konami.masterduel.ach.005	1	2022-03-14 20:22:11	0.33
5	jp.konami.masterduel.ach.006	1	2022-03-07 00:29:47	0.25
6	jp.konami.masterduel.ach.007	1	2022-02-18 19:09:52	0.21
7	jp.konami.masterduel.ach.008	1	2022-03-14 00:56:00	0.20
8	jp.konami.masterduel.ach.009	1	2022-03-13 15:13:22	0.17
9	jp.konami.masterduel.ach.010	1	2022-02-19 19:45:09	0.07
10	jp.konami.masterduel.ach.011	1	2022-03-14 22:58:30	0.07

Dai dati ottenuti definiamo una funzione `platinum` che darà in output True se il gioco è stato platinato. Un gioco si definisce **platinato** se tutti gli achievements sono stati sbloccati dal giocatore.

In [24]:
```
from statistics import mean

def platinum(x):
    count = 0
    for i in range(len(x)):
        if x.iloc[i]['achieved'] == 1:
            count += 1
    if count == len(x):
        return True
```

```
for i in range(len(achievements)):
    meanAch = mean((achievements[i]['global_percentage_of_unlock'])) * 100
    meanAch = str("{:.2f}".format(meanAch)) + '%'
    if platinum(achievements[i]):
        print("Il giocatore ha sbloccato tutti gli achievements di " + gameN
    else:
        print("Il giocatore non ha sbloccato tutti gli achievements di " + g
```

```
Il giocatore ha sbloccato tutti gli achievements di Resident Evil 2. La med
ia di rarita' dei vari trofei e' 41.55%
Il giocatore non ha sbloccato tutti gli achievements di The Binding of Isaa
c. La media di rarita' dei vari trofei e' 16.35%
Il giocatore ha sbloccato tutti gli achievements di Yu-Gi-Oh!  Master Duel.
La media di rarita' dei vari trofei e' 30.82%
```

Calcolare la media di rarità dei vari achievements, cioè la media delle percentuali globali, ci può aiutare nell'individuare la difficoltà di uno specifico gioco. Possiamo dunque affermare che giochi con una media percentuale di sblocco bassa hanno una difficoltà medio/alta, viceversa giochi con una media percentuale di sblocco alta hanno una difficoltà bassa.

Passiamo adesso allo studio dei generi più giocati dall'utente. Visualizzeremo questa informazione con un grafico a torta che renderà immediatamente realizzabile una classifica di preferenze.

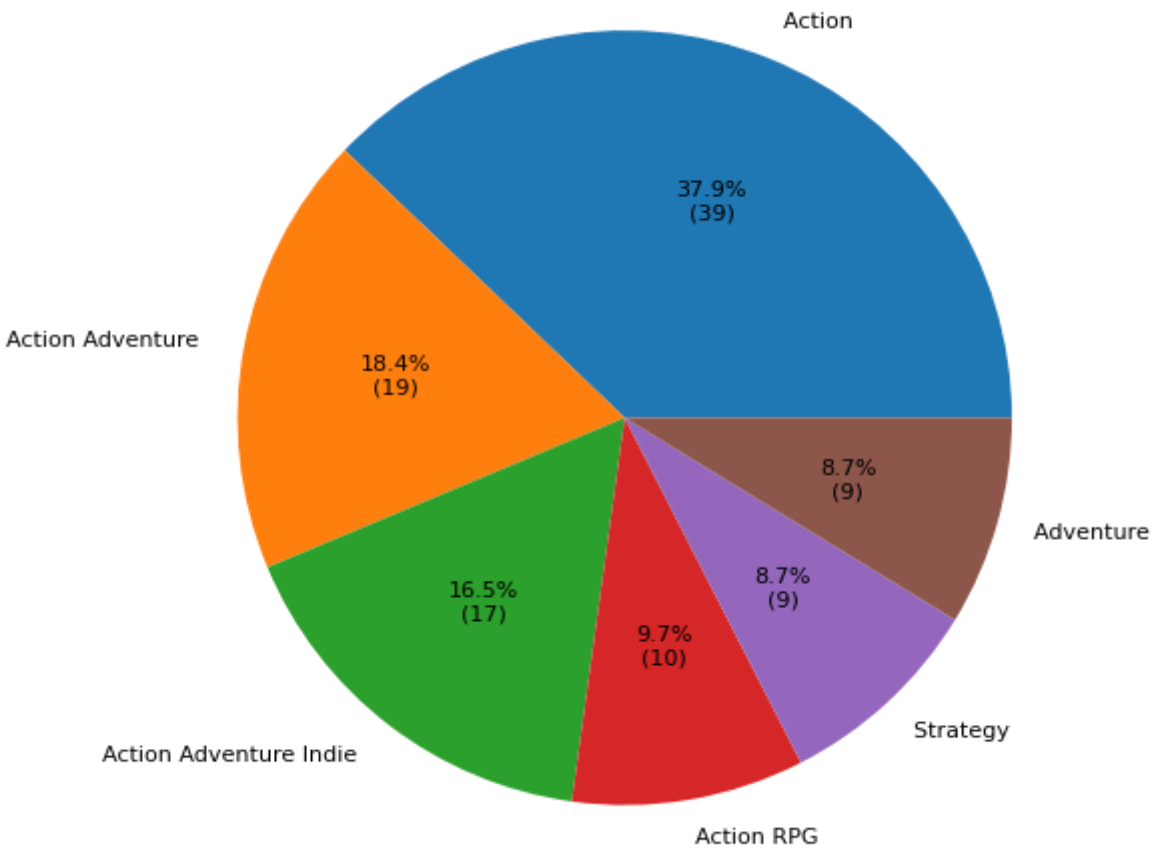
In [25]:
```
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

def autopct_format(values):
    def my_format(pct):
        total = sum(values)
        val = int(round(pct*total/100.0))
        return '{:.1f}%\n({v:d})'.format(pct, v=val)
    return my_format

filtered_data = data[data['genres'] != "NA"] # rimuoviamo i giochi il cui g

figure(figsize = (8, 8), dpi = 80)
genres_dict= filtered_data['genres'].value_counts().to_dict()
genres_list = list(genres_dict.keys())
genres_quantity = list(genres_dict.values())

plt.pie(genres_quantity[:6], labels = genres_list[:6], autopct = autopct_for
plt.show()
```



Considerando i giochi frequentemente giocati dall'utente calcoliamo una stima che descrive quante ore gioca in media l'utente:

In [26]:
```
def avgHoursOfGame():
    result = data.loc[(data['recently_played'] == "yes") & (data['hours_of_g
    result = math.ceil(mean(result['hours_of_game']))
    return str(result)

print("Considerando che 'recently_played' ha al suo interno record che rigua
```

```
Considerando che 'recently_played' ha al suo interno record che riguardan
o le ultime due settimane di gioco, calcoliamo che l'utente gioca in medi
a 57 ore in due settimane. Dunque 57/14, l'utente gioca circa 4 ore al gi
orno.
```

Conclusione

Il nostro studio ha dato come risultato una classifica di generi per l'utente:

1. Al primo posto troviamo i titoli di **Azione**
2. Al secondo posto troviamo i titoli di **Azione** e **Avventura**
3. Al terzo posto troviamo i titoli di **Azione**, **Avventura** e **Indie**.

Da cui risulta evidente che il genere preferito dell'utente sia **Azione**. Inoltre abbiamo concluso, sulla base delle ore di gioco totali riferite ai titoli giocati nelle ultime due

settimane, che l'utente gioca in media 4 ore al giorno.

Questo risulta dunque un caso di studio base possibile attraverso l'utilizzo dei metodi pubblici forniti da Steam e applicabile solo esclusivamente a profili **pubblici**. È possibile ampliare lo studio anche su profili **privati** grazie al rilascio del `clientId` da parte di Valve con un'apposita richiesta formale inviata all'azienda stessa. L'uso generale dell'API rimane comunque abbastanza semplice e non vi è strettamente bisogno di un wrapper per poter semplificare le operazioni.

Fonti:

1. https://it.wikipedia.org/wiki/Steam_(informatica)
2. https://steamcommunity.com/dev?l=italian