

```

PPOptional.add_argument('-v', dest='verbose', action='count', default=0, help='increase verbosity level')
PPOptional.add_argument('--version', version=Version('1.0.0'), help='laZagne version')

PWrite = argparse.ArgumentParser(add_help=False, formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
PWrite._optionals.title = 'output'
PWrite.add_argument('-w', dest='write', action='store_true', help='write a text file on the current directory')

all_subparser = []
for c in category.keys():
    category[c]['parser'] = argparse.ArgumentParser(add_help=False, formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
    category[c]['parser']._optionals.title = category[c]['help']

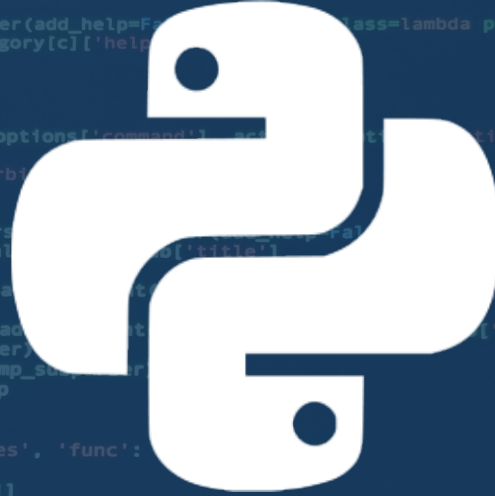
    category[c]['subparser'] = []
    for module in modules[c].keys():
        m = modules[c][module]
        category[c]['parser'].add_argument(m.options['command'], dest=m.options['dest'], action=m.options['action'], help=m.options['help'])

        if m.suboptions and m.name != 'thunderbolt':
            tmp = []
            for sub in m.suboptions:
                tmp_subparser = argparse.ArgumentParser(add_help=False, formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
                tmp_subparser._optionals.title = sub['title']
                if 'type' in sub:
                    tmp_subparser.add_argument(sub['command'], dest=sub['dest'], action=sub['action'], help=sub['help'])
                else:
                    tmp_subparser.add_argument(sub['command'], dest=sub['dest'], help=sub['help'])
                tmp.append(tmp_subparser)
            all_subparser.append(tmp_subparser)
            category[c]['subparser'] += tmp

parents = [PPOptional] + all_subparser + [PWrite]
dic = {'all': {'parents': parents, 'help': 'Run all modules', 'func': 'run_all'}}
for c in category.keys():
    parser_tab = [PPOptional, category[c]['parser']]
    if 'subparser' in category[c]:
        if category[c]['subparser']:
            parser_tab += category[c]['subparser']
    parser_tab += [PWrite]
    dic_tmp = {c: {'parents': parser_tab, 'help': 'Run %s module %s' % (c, 'run %s module')}}
    dic = dict(dic.items() + dic_tmp.items())

subparsers = parser.add_subparsers(help='Choose a main command')
for d in dic.keys():
    subparsers.add_parser(d, parents=dic[d]['parents'], help=dic[d]['help']).set_defaults(func=dic[d]['func'], auditType=d)

```



# LISTAS

## PROGRAMACIÓN DE COMPUTADORAS III

Abdel G. Martínez L.

# CONCEPTO Y DEFINICIÓN

---

- ⦿ Colección de datos ordenados
- ⦿ Contienen cualquier tipo de dato: número, cadena, booleano, listas
- ⦿ Sus valores van separados por coma, entre corchetes:

```
>>> e1 = [45, "lista", False, [100, 101]]
```

# OPERACIÓN: ACCEDER

---

- ⦿ Para acceder a los elementos de una lista se escribe el nombre de la lista y se indica el número del índice entre corchetes:

```
>>> e2 = [True, False]
```

```
>>> val = e2[0]
```

```
>>> print(val)
```

- ⦿ El índice del primer elemento de la lista es siempre 0, no 1

# OPERACIÓN: SLICING

---

- ⦿ Es un mecanismo para seleccionar porciones de una lista
- ⦿ En lugar de un índice, se indica el número de inicio y fin separados por dos puntos:

```
>>> e3 = [1, 2, 3, 4, 5]
```

```
>>> val1 = e3[0:2]
```

```
>>> print(val1)
```

- ⦿ Se omite el número de la posición final

# OPERACIÓN: AGREGAR

---

- Se utiliza la función `append()`

```
>>> e4 = [1, 2, 3]
```

```
>>> e4.append(4)
```

```
>>> print(e4)
```

```
[1, 2, 3, 4]
```

# OPERACIÓN: ELIMINAR

---

- Se utiliza la sentencia `del`

```
>>> e5 = ["matematica", "gramatica", "ciencias"]
```

```
>>> print(e5)
```

```
['matematica', 'gramatica', 'ciencias']
```

```
>>> del e5[0]
```

```
>>> print(e5)
```

```
['gramatica', 'ciencias']
```

# OPERACIÓN: LONGITUD

---

- ⦿ Para buscar el largo de la lista usamos la función `len()`

```
>>> e6 = [5, 10, 15, 20]
```

```
>>> print(len(e6))
```

```
4
```

# OPERACIÓN: CONCATENAR

---

- ⦿ Para concatenar se utiliza el operador aritmético de suma

```
>>> e7 = [1, 2, 3]
```

```
>>> e8 = [4, 5, 6]
```

```
>>> print(e7 + e8)
```

```
[1, 2, 3, 4, 5, 6]
```