

```

PPOptional.add_argument('-v', dest='verbose', action='count', default=0, help='increase verbosity level')
PPOptional.add_argument('--version', version='Version ' + str(constant.CURRENT_VERSION), 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']

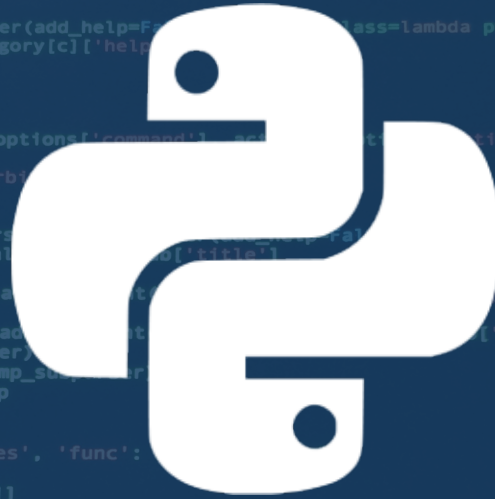
    category[c]['subparser'] = []
    for module in modules[c].keys():
        m = modules[c][module]
        category[c]['parser'].add_argument(m.options['command'], dest=m.options['dest'], action=m.options['action'], help=m.options['help'])

        if m.suboptions and m.name != 'thunderbird':
            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': 'runModules'}}
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' % c, 'func': 'runModule'}}
    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)

```



# TUPLAS

## PROGRAMACIÓN DE COMPUTADORAS III

Abdel G. Martínez L.

# CONCEPTO Y DEFINICIÓN

---

- ⦿ Secuencias de objetos, como las listas
- ⦿ Son inmutables, es decir, no pueden ser cambiadas
- ⦿ Sus valores van separados por coma, entre paréntesis:

```
>>> t1 = (65, "tupla", True)
```

# OPERACIÓN: ACCEDER

---

- ⦿ Se hace igual que en las listas
- ⦿ Para acceder a los elementos de una tupla se escribe el nombre de la tupla y se indica el número del índice entre corchetes:

```
>>> t2 = ("lunes", "miercoles", "viernes")
```

```
>>> print(t2[0])
```

```
lunes
```

```
>>> print(t2[1:3])
```

```
['miercoles', 'viernes']
```

# OPERACIÓN: ACTUALIZAR

---

- ⦿ No es posible actualizar una tupla, son inmutables
- ⦿ Se puede crear una nueva tupla basándose en una tupla existente

```
>>> t3 = (1, 2, 3)
```

```
>>> t4 = t3[0:1]
```

```
>>> print(t4)
```

```
(1,)
```

# OPERACIÓN: ELIMINAR

---

- ⦿ No se puede eliminar un elemento de una tupla
- ⦿ En su lugar, se borra toda la tupla

```
>>> t5 = ("calculo", "ortografia", "civica")
```

```
>>> del t5
```

```
>>> print(t5)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
NameError: name 't5' is not defined
```

# OPERACIÓN: LONGITUD

---

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

```
>>> t6 = ('a', 'b', 'c')
```

```
>>> print(len(t6))
```

```
3
```

# OPERACIÓN: CONCATENAR

---

- ⦿ Para concatenar se utiliza el operador aritmético de suma

```
>>> t7 = (3, 6, 9)
```

```
>>> t8 = (2, 4, 6)
```

```
>>> print(t7 + t8)
```

```
(3, 6, 9, 2, 4, 6)
```