



ENTRADA DE DATOS

PROGRAMACIÓN DE COMPUTADORAS III

Abdel G. Martínez L.

CONCEPTO Y DEFINICIÓN

- ⦿ Existen múltiples formas de generar datos en un programa
- ⦿ Base de datos, fichero, clics del ratón o de Internet
- ⦿ La forma más común es a través del teclado

CONCEPTO Y DEFINICIÓN

- ⦿ La función `input ()` es la solución de Python para esta situación
 - Detiene el flujo de ejecución hasta que el usuario escriba un texto y se presione la tecla ENTER
 - Admite un parámetro, el cual se imprime en pantalla como ayuda para el usuario final

CONSIDERACIONES ESPECIALES

- ⦿ Todo texto de entrada es una cadena
- ⦿ Si se desea transformar a otro tipo de dato, debe utilizarse otros algoritmo y funciones

```

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

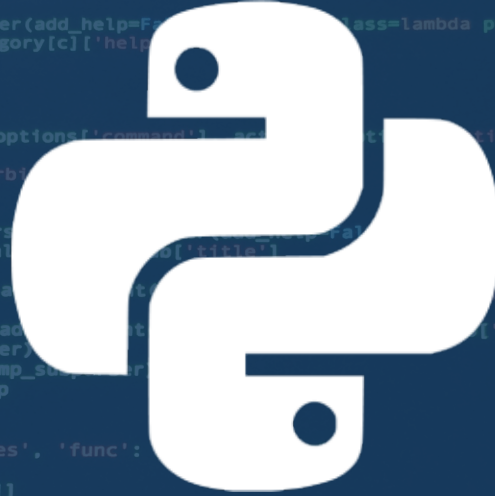
    category[c]['subparser'] = []
    for module in modules[c].keys():
        m = modules[c][module]
        category[c]['parser'].add_argument(m.options['command'], dest=m.options['dest'], help=m.options['help'])

        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'], help=sub['help'], action=sub['action'], dest=sub['dest'], help=sub['help'])
                else:
                    tmp_subparser.add_argument(sub['command'], dest=sub['dest'], help=sub['help'], action=sub['action'], 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_modules}}
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' % c, 'func': run_module}}
    dic = dict(dic.items() + dic_tmp.items())

subparsers = parser.add_subparsers(help='laZagne subcommands')
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)

```



SALIDA DE DATOS

PROGRAMACIÓN DE COMPUTADORAS III

Abdel G. Martínez L.

CONCEPTO Y DEFINICIÓN

- ⦿ Cada programa debe poder comunicarse con su entorno
- ⦿ Para este propósito nacieron las funciones de entrada/salida
- ⦿ En el video anterior aprendieron cómo ingresar datos
- ⦿ Imprimiremos los resultados de nuestros programas en pantalla

CONCEPTO Y DEFINICIÓN

- ⦿ La función `print()` es la solución de Python para esta situación

- Permite imprimir directamente un texto

```
>>> print("Hola Eduvolucion!")
```

```
Hola Eduvolucion!
```

- Puede imprimir el valor de una variable

```
>>> x = 3
```

```
>>> print(x)
```

```
3
```

CONCEPTO Y DEFINICIÓN

⦿ La función `print()` es la solución de Python para esta situación

- Podemos imprimir varios valores separados por coma

```
>>> a = 5
```

```
>>> b = 7
```

```
>>> print(a, b)
```

- Permite mezclar impresión de textos y valores de variables

```
>>> nombre = "Carlos"
```

```
>>> print("Tu nombre es " + nombre)
```


CONSIDERACIONES ESPECIALES

- ⦿ Si la variable es de tipo numérica, debemos utilizar la función `str()` para convertir su valor en una cadena

```
>>> edad = 35
```

```
>>> print("Tu edad es " + str(edad))
```



DECISIONES

PROGRAMACIÓN DE COMPUTADORAS III

Abdel G. Martínez L.

CONCEPTO Y DEFINICIÓN

- ⦿ Las decisiones permiten comprobar condiciones
- ⦿ Hacen que nuestro programa se comporte de diferentes maneras dependiendo de la condición
- ⦿ La sentencia `if` es utilizada como condicional
- ⦿ Evalúa una condición, si es verdadera ejecuta un código; si es falsa, ejecuta otro código o continúa la ejecución del programa
- ⦿ Utiliza valores booleanos (`True`, `False`)

ESTRUCTURA

- ⦿ La primera línea contiene la expresión lógica a evaluar
- ⦿ Debe terminar siempre con dos puntos (:)
- ⦿ Luego van las órdenes a ejecutar cuando se cumple la condición
- ⦿ Estas órdenes deben ir en bloque de sangrado

```
>>> num = input("Escriba un número: ")
```

```
>>> if int(num) < 0:
```

```
...     print("El número es negativo")
```

```
>>> print("Su número es " + num)
```

TIPOS

⦿ *Simple*

```
>>> if 2 < 0:  
...     print("El número es negativo")
```

⦿ *Bifurcado*

```
>>> if int(num) < 0:  
...     print("El número es negativo")  
  
>>> else:  
...     print("El número es positivo")
```

TIPOS

◉ *Múltiple*

```
>>> if int(num) < 0:
...     print("El número es negativo")
... elif int(num) == 0:
...     print("El número es cero")
... else:
...     print("El número es negativo")
>>>
```

TIPOS

◉ *Anidado*

```
>>> if int(intento) == 5:
...     print("Ganastes")
... else:
...     if int(intento) < 5:
...         print("Un poco más alto")
...     else:
...         print("Un poco más bajo")
```



CICLOS

PROGRAMACIÓN DE COMPUTADORAS III

Abdel G. Martínez L.

CONCEPTO Y DEFINICIÓN

- ⦿ Un ciclo es una secuencia de instrucciones que se van a repetir continuamente hasta que se cumpla cierta condición
- ⦿ Debemos evitar los ciclos infinitos, los cuales son ciclos que carecen de una condición válida de salida
- ⦿ Cada pase a través del ciclo se conoce como iteración
- ⦿ Existe varios tipos de ciclos: controlados, rangos numéricos y el basado en un iterador.

CATEGORÍAS

⦿ *While*

- Ejecuta una sentencia hasta que se cumpla una condición

```
>>> x = 0
```

```
>>> while x < 3:
```

```
...     print("Sigo en el ciclo")
```

```
...     x = x + 1
```

```
...
```

```
>>> print("Hasta luego")
```

CATEGORÍAS

⦿ *While-Else*

- Las sentencias del `else` se ejecuta después de salir del ciclo

```
>>> x = 0
```

```
>>> while x < 3:
```

```
...     print("Sigo en el ciclo")
```

```
...     x = x + 1
```

```
... else:
```

```
...     print("Hasta luego")
```

CATEGORÍAS

⦿ *For*

- Ejecuta una sentencia un número específico de veces

```
>>> languages = ["C", "C++", "Perl", "Python"]
```

```
>>> for l in languages:
```

```
...     print(l)
```

```
...
```

```
>>>
```

CATEGORÍAS

⦿ *For*

- Puede utilizar la función `range()` para manejarse por rangos

```
>>> for x in range(5):
```

```
...     print(x)
```

```
...
```

```
>>>
```

CATEGORÍAS

⦿ *For-Else*

- Las sentencias del `else` se ejecuta después de salir del ciclo

```
>>> languages = ["C", "C++", "Perl", "Python"]
```

```
>>> for l in languages:
```

```
...     print(l)
```

```
... else:
```

```
...     print("Se acabaron")
```

```
...
```

CONSIDERACIONES ESPECIALES

- ⦿ La sentencia `break` rompe el ciclo en ejecución
- ⦿ La sentencia `pass` no hace ninguna acción
- ⦿ La sentencia `else` no se ejecuta luego de un `break`