
UAP

UAP | UNIVERSIDAD ADVENTISTA DEL PLATA

ALGORITMOS Y ESTRUCTURAS DE DATOS

FUNCIONES - PYTHON

CULTURA

SAWABONA - SHIKOBA

Costumbre de saludo de tribu del sur de África.

Reconstruye la comunicación emocional.

https://economiahumana.org/sawabona-shikoba-creando-comunidad-desde-el-amor-y-el-respeto/

Principales conceptos

CONCEPTO

Mutabilidad

La mutabilidad define si un dato puede ser cambiado tras ser inicializado o si siempre mantiene el mismo valor, tanto para la variable original como para las demás referencias que tenga valor.

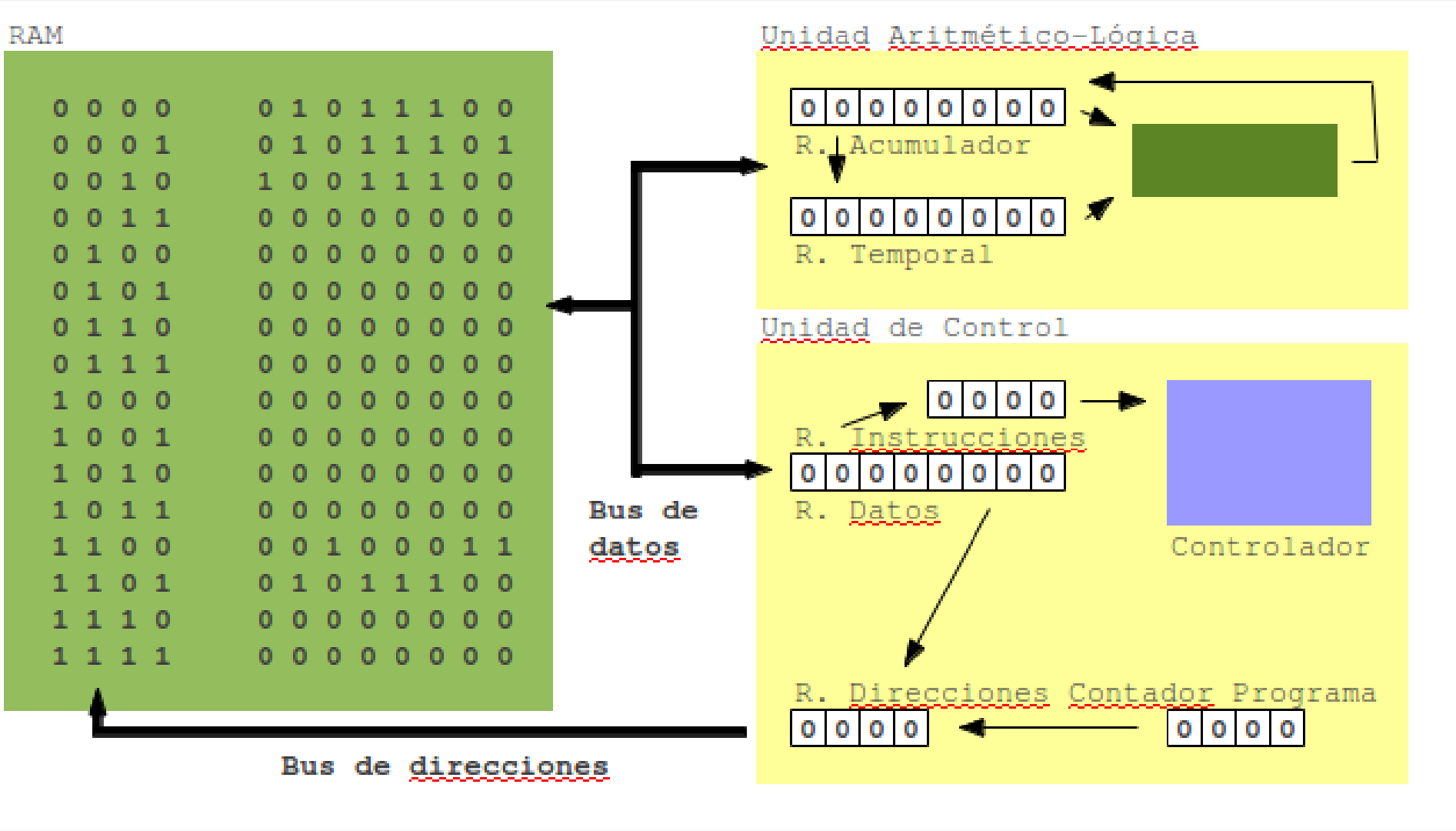
```python
a = 3
a = 4
print(a)
#4

x = ("uno", "dos", "tres")
print(x[1])
#dos

x[1] = "cero"
#TypeError: 'tuple' object does not support item assignment
```

CONCEPTO

PUNTERO

Variable cuyo contenido es la dirección de memoria de otra variable

Variable



https://castilloinformatica.es/wiki/index.php?title=Secuencia_de_ejecuci%C3%B3n_de_una_instrucci%C3%B3n

```python
a = ["uno", "dos"]
print(id(a))
#139942723863872

b = a
x = a.copy()

print(id(b))
#139942723863872

a[0] = "tres"
print(b)
#['tres', 'dos']

print(id(x))
#140311799231744
print(x)
#['uno', 'dos']
```

CONCEPTO

COPY()

Clona un objeto en otro, pero es una copia superficial.

```python
import copy

x = [[1,2,3], ['Juan', 'Ana'], True]
y = x.copy()
z = copy.deepcopy(x)

print(x)
#[[1, 2, 3], ['Juan', 'Ana'], True]
print(y)
#[[1, 2, 3], ['Juan', 'Ana'], True]

x[0][1] = 333
print(x)
#[[1, 333, 3], ['Juan', 'Ana'], True]
print(y)
#[[1, 333, 3], ['Juan', 'Ana'], True]

print(z)
#[[1, 2, 3], ['Juan', 'Ana'], True]
```

CONCEPTO

DEEPCOPY()

Clona un objeto en otro de forma completa

Requiere importar funcionalidades extras

CONCEPTO

ASIGNACIONES SIMPLES Y MULTIPLES

Una de las operaciones más usadas en algoritmos.

Permite la inicialización o redefinición del valor de una variable

```python
a = b = c = 1

print(a, b, c)
#1, 1, 1

b = 3
print(a, b, c)
#1, 3, 1
```

```python
a, b, c = [1, 2, 3]
print(a, b, c)
#1, 2, 3

d, e, f = 'ABC'
print(d, e, f)
#A B C

g, *h, i = (1, 2, 3, 4, 5, 6)
print(g, h, i)
#1 [2, 3, 4, 5] 6
```

CONCEPTO

OPERADOR WALRUS

Desde Python 3.8 es la forma de asignar cuando se usa dentro de una expresión. Mejora la visibilidad.

```
1 a = 0
2
3 if (a := 1) == 1:
4     print("yeap")
5 else:
6     print("nop")
```

CONCEPTO

FUNCIONES

Subprograma o subrutina que forma parte de un algoritmo principal.

Procedimientos VS funciones.

Ayuda a organizar mejor el código.

Permite la reutilización de porciones de código y evita repetirse. Facilita el mantenimiento y testeo (divide y conquistarás)

```
1  def nombre_funcion():
2      print("Aca el bloque de código")
3      print("Prestar atención a la indentación")
4      return "Resultado es opcional"
5
6  print(nombre_funcion())
7  """
8  Aca el bloque de código
9  Prestar atención a la indentación
10 Resultado es opcional
11 """
```

CONCEPTO

FUNCIONES: Parámetros y argumentos

Es el mecanismo para la introdución de variables o valores del exterior de la función.

Los nombres de variables utilizados en la definición de la función son los parámetros.

Las variables que portal los valores desde el exterior al utilizar la función son los argumentos.

Todos los argumentos son pasados a la función por referencia.

```
13 a = [1, "algo"]
14
15
16 def test(x):
17     print(id(x))
18     x[0] = 4
19
20
21 print(id(a))
22 test(a)
23 print(a)
24
25 """
26 139844042855680
27 139844042855680
28 [4, 'algo']
29 """
```

CONCEPTO

FUNCIONES: Parámetros y argumentos

En la utilización de la función, los argumentos son pasados en el mismo orden que fueron definidos los parámetros o utilizando los nombres definidos internamente

```python
apellido, nombre = ["Perez", "Juan"]

def nombre_completo(a, n):
    return a + ", " + n

print(nombre_completo(apellido, nombre))
#Perez, Juan
```

```python
apellido, nombre = ["Perez", "Juan"]

def nombre_completo(a, n):
    return a + ", " + n

print(nombre_completo(n=nombre, a=apellido))
#Perez, Juan
```

CONCEPTO

FUNCIONES: Parámetros y argumentos

Parámetros pueden tener valor por defecto si al invocar la función no se le pasan argumentos.

```python
apellido, nombre = ["Perez", "Juan"]

def nombre_completo(a="Gonzales", n="Mario"):
    return a + ", " + n

print(nombre_completo(n=nombre, a=apellido))
print(nombre_completo())
#Perez, Juan
#Gonzales, Mario
```

CONCEPTO

FUNCIONES: args y kwargs

Mecanismos para pasar una cantidad dinámica de argumentos a una fución.

Si son solo argumentos posicionales (en forma de lista simple) se usa *

Si son argumentos del tipo clave-valor se usa **

```python
def mi_funcion(*args, **kwargs):
    print(f'Argumentos simples: {args} - Argumentos k-v {kwargs}')

mi_funcion()
#Argumentos simples: () - Argumentos k-v {}

mi_funcion(66)
#Argumentos simples: (66,) - Argumentos k-v {}

mi_funcion(marca='Ford')
#Argumentos simples: () - Argumentos k-v {'marca': 'Ford'}


mi_funcion(13, 'algo', modelo='TNT300')
#Argumentos simples: (13, 'algo') - Argumentos k-v {'modelo': 'TNT300'}
```

CONCEPTO

FUNCIONES: Tipado

Como python es tipado dinámico, se definen los tipos de dato de los parámetros al momento de la asignación en tiempo de ejecución.

Si queremos forzar o restringir el tipo de dato a aceptar en la función se debe especificar.

```python
def ejemplo(x: int, y: str="nada") -> bool:
    print(x)
    print(y)
    return True


ejemplo(1)
#1
#nada

ejemplo(33, "algo")
#33
#algo

x = ejemplo(0)
print(type(x))
#<class 'bool'>
```

CONCEPTO

FUNCIONES: Documentación

Sistema para documentar funciones. Consiste en utilizar un comentario de bloque inmediatamente despues de la definición y dentro de la misma.

```python
def ejemplo(x: int, y: str="nada") -> bool:
    """Función de ejemplo de uso de parámetros con tipos de datos definidos
    :param x: un parámetro entero
    :param y: un parámetro string con valor por defecto
    :return bool: valor de retorno es ...
    """
    print(x)
    print(y)
    return True

help(ejemplo)
"""
Help on function ejemplo in module __main__:

ejemplo(x: int, y: str = 'nada') -> bool
    Función de ejemplo de uso de parámetros con tipos de datos definidos
    :param x: un parámetro entero
    :param y: un parámetro string con valor por defecto
    :return bool: valor de retorno es ...
"""
```

CONCEPTO

ÁMBITO O CONTEXTO DE UNA VARIABLE (scope)

Una variables es global cuando puede ser accedida desde cualquier parte del programa. Será local cuando solo pueda ser accedida en el contexto donde fue definida.

Dependiendo dónde fueron declaradas o inicializadas serán variables globales o locales.

```python
z = 12


def test():
    z = 33
    print(z)


test()
#33
print(z)
#12
```

```
1  z = 12
2  x = "global"
3
4  def test():
5      z = 33
6      print(x)
7
8
9  test()
10 #global
11 print(x)
12 #global
```

```python
z = 12
x = "global"

def test():
    z = 33
    global x
    print(x)
    x = "cambio global"



test()
#global
print(x)
#cambio global
```

CONCEPTO

EXCEPCIONES

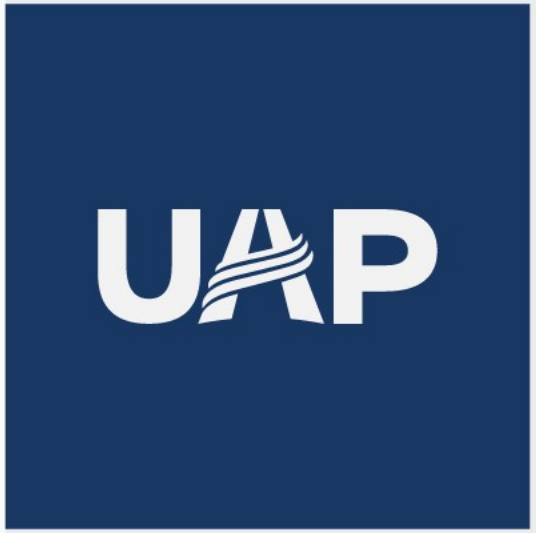
Gestión de casos anómalos o problemas inesperados.

Por ejemplo división entre 0 o acceso a variable no definida (null pointer)

Se producen de forma automática o manual.

```python
try:
    a = 5 / 0
except Exception as e:
    print(f'Error inesperado "{e}"')

print("Programa continua y no se detiene")
#Error inesperado "division by zero"
#Programa continua y no se detiene
```

```python
try:
    raise Exception("Excepción personalizada!")
except Exception as e:
    print(f'Error inesperado "{e}"')

print("Programa continua y no se detiene")
#Error inesperado "Excepción personalizada!"
#Programa continua y no se detiene
```



UAP.EDU.AR | @UAPARGENTINA