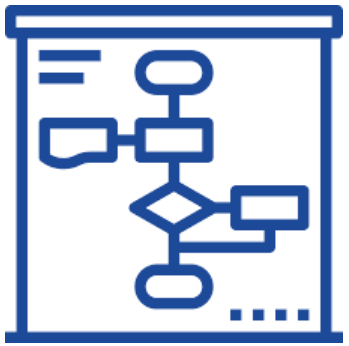
INTRODUCCIÓN A LA PROGRAMACIÓN



OTOÑO, 2022

UNIVERSIDAD TECNOLÓGICA DE CHILE
INSTITUTO PROFESIONAL
CENTRO DE FORMACIÓN TÉCNICA
nacap

UNIDAD II

Lenguajes Altamente Dinámicos/ Débilmente Tipados

Fundamentos Python

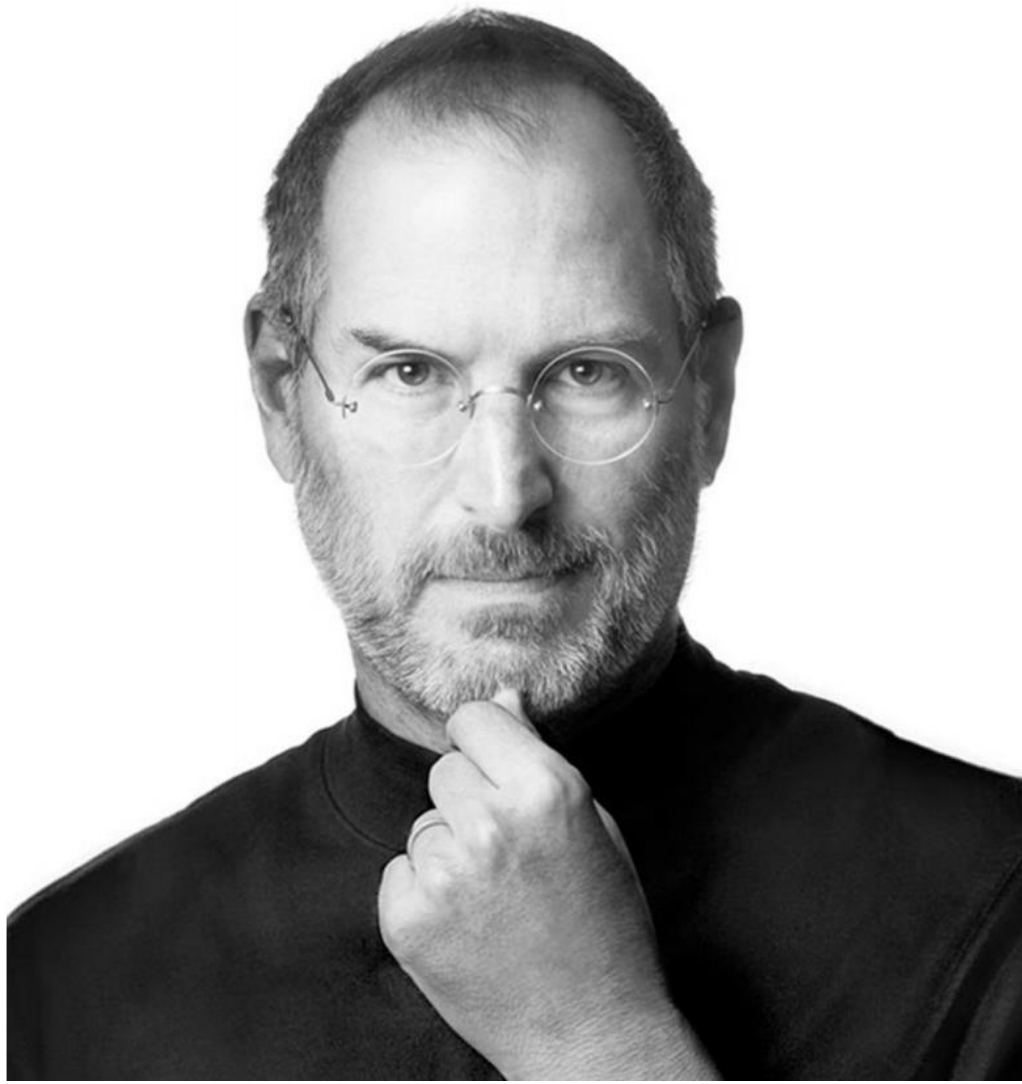
nacap

Motivación

Reflexión

Una posible respuesta que nos hace sentido, es que la programación es útil en diversas áreas del conocimiento, desarrolla el pensamiento lógico y potencia una de las habilidades fundamentales que es la resolución de problemas.

> **"Todo el mundo debería aprender a programar un ordenador, porque te enseña a pensar"**
>
> Steve Jobs



Programación en el mundo laboral y cómo debe convertirse en un hábito

Objetivo	Enfoque	Rol del trabajador	Procesamiento de datos
El objetivo es tener **herramientas de programación** que nos permitan resolver problemas.	**Procesamiento de datos**, ya que este es un problema que se repite en la vida cotidiana de cualquier trabajador.	Todo trabajador debería ser capaz de **manejar herramientas básicas de programación** para poder facilitar sus tareas del día a día.	Especialmente en el ámbito del procesamiento de datos ya que es y será uno de los grandes desafíos del futuro

Bases del lenguaje

Palabras reservadas

Python tiene ciertas palabras reservadas, es decir, que no pueden asignarse a constantes o variables, ya que representan funciones específicas para este lenguaje, por ejemplo:

False	None	True	and	as	assert
async	await	break	class	continue	def
del	elif	else	except	finally	for
from	global	if	import	in	is
in	lambda	nonlocal	not	or	pass
raise	return	try	while	with	yield

```
>>> help("keywords")

Here is a list of the Python keywords.  Enter any keyword to get more help.

and                 elif                if                  print
as                  else                import              raise
assert              except              in                  return
break               exec                is                  try
class               finally             lambda              while
continue            for                 not                 with
def                 from                or                  yield
del                 global              pass
```

```
1   print("Hola Mundo")
```

Nota: Ya analizaremos en detalle la función print().

Sintaxis

- Python fue diseñado para facilitar la lectura y tiene algunas similitudes con el idioma inglés con influencia de las matemáticas.
- Usa nuevas líneas para completar un comando, a diferencia de otros lenguajes de programación que a menudo usan punto y coma o paréntesis.
- Se basa en la sangría, usando espacios en blanco, para definir el alcance; como el alcance de los bucles, funciones y clases. Otros lenguajes de programación a menudo usan corchetes para este propósito.

Bases del lenguaje

Sangría en Python (indentación)

La sangría se refiere a los espacios al comienzo de una línea de código. Mientras que en otros lenguajes de programación la sangría en el código es solo para legibilidad, la sangría en Python es muy importante. Python usa sangría para indicar un bloque de código,

```
4    if 5>2:
5        print("Cinco es mayor que dos")
```

```
4    if 5>2:
5    print("Cinco es mayor que dos")
```

Python le dará un error si omite la sangría:

```
IndentationError: expected an indented block
```

Error Tabulación: Se esperaba un bloque tabulado.

El número de espacios depende del programador, pero debe ser al menos uno.

```
4    if 5 > 2:
5        print("Cinco es mayor que dos")
6    if 5 > 2:
7            print("Cinco es mayor que dos")
```

Se debe usar la misma cantidad de espacios en el mismo bloque de código, de lo contrario Python te dará un error:

```
4    if 5 > 2:
5        print("Cinco es mayor que dos")
6            print("Cinco es mayor que dos")
```

```
1    print("Hola Mundo")
2        print("Hola Mundo")
```

Bases del lenguaje

Comentarios en Python

Los comentarios se pueden utilizar para:

- Explicar el código Python.
- Hacer que el código sea más legible.
- Evitar la ejecución al probar el código.

Los comentarios comienzan con un # y Python no ejecutará dicha línea de comando.

```
1   #print("Hola Mundo")
2   help("if")
```

Los comentarios se pueden colocar al final de una línea y Python ignorará el resto de la línea:

```
1   #print("Hola Mundo")
2   help("if") #Esta función invoca el sistema de ayuda integrado de Python.
```

Para comentarios de varias líneas, dado que Python ignorará los literales de cadena que no están asignados a una variable, puede agregar una cadena de varias líneas (comillas triples) en su código y colocar su comentario

```
4   """ Este es un comentario del
5   código Python, escrito en
6   más de una línea """
7   print("Hola Mundo")
```

Variables

Los nombres de las variables pueden ser letras y números, pero no pueden comenzar con un número. También, puede utilizarse el guion bajo (_), que usualmente se utiliza para nombres largos, por ejemplo, **esto_podria_ser_un_nombre_de_variable_en_python**. La siguiente imagen muestra algunas declaraciones de variables:

```
1   # Estos son algunos nombres de variables.
2   numero_entero = 25
3   numero_decimal_2 = 14.87
4   asignatura = "Algoritmos y programación"
5   nota_para_aprobar = 4.0
6   pi = 3.14
7   aprobado = True
```

Nota: Python no tiene ningún comando para declarar una variable. Una variable se crea en el momento en que le asigna un valor por primera vez, declarando de forma explícita.

Case-Sensitive

Distingue mayúsculas y minúsculas.
Los nombres de las variables distinguen entre mayúsculas y minúsculas.

```
13   a=5
14   A='Programación'
15   #A no sobreescribe a
```

Reglas para las variables

Una variable puede tener un nombre corto (como x e y) o un nombre más descriptivo (edad, nombre del coche, volumen_total). Reglas para variables de Python:

- El nombre de una variable debe comenzar con una letra o el carácter de subrayado.
- El nombre de una variable no puede comenzar con un número.
- El nombre de una variable solo puede contener caracteres alfanuméricos y guiones bajos (A-z, 0-9 y _).
- Los nombres de las variables distinguen entre mayúsculas y minúsculas (la edad, la Edad y la EDAD son tres variables diferentes).

```
1   mivar="Introducción"
2   mi_var="a la"
3   _mi_var="Programación"
4   miVar="Semestre"
5   MYMAR="Otoño"
6   mivar2="2021"
```

```
1   2mivar="Introducción"
2   mi-var="a la"
3   mi var="Programación"
```

Indague sobre las técnicas Camel Case, Pascal Case, Snake Case para la definición de variables de palabras múltiples.

Variables (otros ejemplos)

Asignar valores a múltiples variables en una línea

```
x,y,z="Programación","en","Python"
print(x)
print(y)
print(z)
```

Asignar el mismo valor a múltiples variables en una línea

```
x=y=z="Programación"
print(x)
print(y)
print(z)
```

Tipos de datos

En programación, el tipo de datos es un concepto importante. Las variables pueden almacenar datos de diferentes tipos y diferentes tipos pueden hacer cosas diferentes. Python tiene los siguientes tipos de datos integrados de forma predeterminada, en estas categorías:

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`

Si desea obtener el tipo de datos de una variable, debe utilizar la función **type()**.

```
13   x = 5
14   y = "Programación"
15   print(type(x))
16   print(type(y))
```

```
<class 'int'>
<class 'str'>
```

Si desea especificar el tipo de datos de una variable, puede hacerlo con conversión.

```
15   w = str(3)    # w será '3'
16   x = int(3)    # x será 3
17   y = float(3)  # y será 3.0
18   z = bool(1)   # z será True
```

Bases del lenguaje

Ejemplos tipos de datos

int (entero)

```
x = 1
y = 35656222554887711
z = -3255522
```

float (flotantes)

```
x = 1.10
y = 1.0
z = -35.59
```

float (fcinetpifico)

```
x = 35e3
y = 12E4
z = -87.7e100
```

bool (booleano)

```
x = True
```

str (string)

```
x = "Hola Mundo"
```

Conversión de tipo: Puede convertir de un tipo a otro con los métodos str(), int (), float () , complex ().

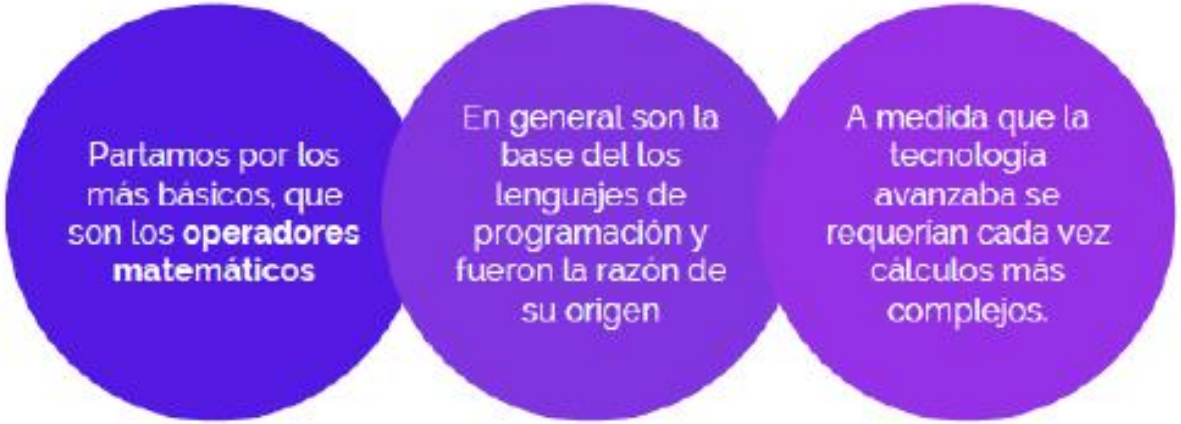
```
#convert from int to float:
x = float(1)

#convert from float to int:
y = int(2.8)

#convert from int to complex:
z = complex(x)

print(x)
print(y)
print(z)

print(type(x))
print(type(y))
print(type(z))
```

Salida

```
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

Recordando...

Operadores y Funciones



La jerarquía de los operadores es igual a la del álgebra, aunque puede alterarse mediante el uso de paréntesis.

Para obtener la raíz cuadra puede utilizar:
*número**(1/2)*
[más adelante lo realizaremos por función]

+	Suma los valores de la izquierda y la derecha

```
(1001+817)
```

−	Resta el valor de la derecha al valor de la izquierda

```
(7-4)
```

*	Multiplica el valor de la izquierda por el de la derecha

```
(5*8)
```

/	Divide el valor de la izquierda por el de la derecha

```
(12/4)
```

**	Calcula la potencia. El valor de la izquierda es la base y el de la derecha es el exponente.

```
(2**5)
```

//	Divide el valor de la izquierda por el de la derecha, y entrega el valor entero del resultado.

```
(5//4)
```

%	Divide el valor de la izquierda por el de la derecha. No entrega el resultado sino que el resto de esta división.

```
(7%5)
```

Recordando...

Operadores y Funciones (consideraciones)

Cómo redondear un número en Python

Redondear un número en Python utilizando el método estándar matemático, es decir:

- De 0.1 a 0.4 decimal, se redondea al entero inferior
- De 0.5 a 0.9 decimal, se redondea al entero superior

```
round(3.2)
[Out] 3

round(3.8)
[Out] 4
```

Cómo truncar un número en Python

Truncar un número en Python. Para truncar un número podemos hacerlo de dos formas, utilizando la función del módulo **math** (**math.trun**) o bien utilizando la función predefinida ***int***

```
int(3.9)
[Out] 3

math.trunc(3.9)
[Out] 3

int(-3.9)
[Out] -3

math.trunc(-3.9)
[Out] -3
```

Recordando...

Operadores lógicos de comparación

igual	==	Compara el valor de la derecha y la izquierda	(10==9)
distinto	!=	Compara el valor de la derecha y la izquierda	(10!=9)
menor que	<	Menor que	(11<7)
mayor que	>	Mayor que	(7>7)
mayor o igual que	>=	Mayor o igual que	(7>=7)
menor o igual que	<=	Menor o igual que	(7<=7)

Los operadores lógicos son usados para operaciones de álgebra Booleana, es decir, para describir relaciones lógicas, expresadas como verdadero o falso.

Recordando...

Operadores lógicos binarios

Símbolo	Significado	Descripción	Ejemplo
AND	Y (Conjunción)	Devuelve verdadero solo cuando ambas proposiciones son verdaderas	(1 = 1) AND (2*2 = 4) es Verdadero
OR	O (Disyunción)	Devuelve verdadero cuando al menos una de las proposiciones es verdadera.	(2=2) OR (2*2=7) es Verdadero

NOT
(negación)

El operador lógico `not (!)` sirve para poder "cambiar" el valor de una operación lógica. Es decir, si uno le aplica este operador a una operación lógica, cambia el valor de VERDADERO a FALSO o viceversa.

Ejemplos

```
x < 5 and  x < 10

x < 5 or x < 4

not(x < 5 and x < 10)
```

Operadores de asignación de Python

Operador	Ejemplo	Igual que
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

Leer y Escribir

Para leer un dato que ingresa el usuario, podemos usar la función **input()** y desplegar un mensaje con **print()**, como se muestra en el siguiente ejemplo:

```
1   # Primer programa: Saludo
2
3   # leer entrada y guardar en variable "nombre"
4   nombre = input("Ingrese su nombre: ")
5   # mostrar saludo
6   saludo = "Hola " + nombre + ", bienvenido/a a Algoritmos y Programación"
7   print(saludo)
```

Nota: En este ejemplo se utiliza el operador **+** para concatenar cadenas de string.
También es posible utilizar el operador * para obtener cadenas de caracteres.

```
print("*" * 5)
```

Salida `*****`

Consideraciones Leer y Escribir

- Durante el ingreso, podemos asignar un tipo a la variable:

```
precio_dolar=int(input("Ingresa el precio del dolar:\n"))
cantidad_de_dolares=int(input("Ingresa la cantidad de dolares por cambiar:\n"))
cantidad_en_pesos=precio_dolar*cantidad_de_dolares
print(cantidad_en_pesos)
```

- La invocación de print(). genera una línea vacía (esta interpretación también es correcta) su salida es solo una nueva línea.

```
1   print("Introducción a la Programación")
2   print()
3   print("Otoño 2021")
```

- Caracteres de escape y nueva línea:
  La barra invertida (\) tiene un significado muy especial cuando se usa dentro de las cadenas, es llamado el carácter de escape.

txt = "We are the so-called \"Vikings\" from the north."

```
1   print("La Witsi Witsi Araña\nsubió a su telaraña.\n")
2   print()
3   print("Vino la lluvia\ny se la llevó.")
```

La letra n colocada después de la barra invertida proviene de la palabra newline (nueva línea).

Consideraciones Leer

- Python asume que cualquier dato que el usuario ingresa es de tipo string, como muestra el siguiente ejemplo:

```
x = input() #Se ingresa 5 en x
y = input() #Se ingresa 6 en y
print(x+y)
```

Salida

```
5
6
56
```

Si necesitamos realizar alguna operación aritmética con los valores de x e y, debemos considerar ingresarlos con el tipo adecuado o realizar la conversión.

Consideraciones Escribir

- Los argumentos de palabras claves:
  end=" " , palabra clave que identifica el argumento (end: termina aquí);
  un signo de igual (=); y un valor asignado a ese argumento.

```
print("Mi nombre es ", end="")
print("Monty Python.")
```

Salida `Mi nombre es Monty Python.`

- sep=" " , palabra clave que identifica el argumento (sep: separador); un signo de igual (=); y un valor asignado a ese argumento.

```
print("Mi", "nombre", "es", "Monty", "Python.", sep="-")
```

Salida `Mi-nombre-es-Monty-Python.`

- Ambos argumentos de palabras clave pueden mezclarse en una invocación:

```
print("Mi", "nombre", "es", sep="_", end="*")
print("Monty", "Python.", sep="*", end="*\n")
```

Salida `Mi_nombre_es*Monty*Python.*`

inacap.cl