#20_Type_Conversions

04/06/2025

- Type coercion is the process of converting a value of one type to another.

Explicit Type Coercion

- Intentional implementation of built-in functions to convert a value of one type to another.
  e.g.; using `int` and `float` to convert a string into a numerical value before performing arithmetic

Coercing values to Integers

The `int()` function is used to convert values to integers

```
integer_str = "42"
integer = int(integer_str)
print(integer) # Output: 42
```

Cannot convert a non-numeric string to an integer. It will result in a `ValueError`

```
non_numeric_str = "abc"
try:
    non_numeric_str = int(non_numeric_str)
except ValueError:
    print("Cannot convert to int.")
```

- Often you pass a numeric string to `int()` function but you can also pass real (floating-point) numbers to `float()` function.

04/06/2025

- Passing any other data-types, like a list, will result in a TypeError

```
input_value = ['42']
try:
    numeric_value = int(input_value)
except TypeError:
    print("Invalid input type for conversion.")
```

Coercing values to floats

- The float() function converts values to floating-point numbers

```
float_str = "3.14"
float_number = float(float_number)
print(float_number) # Output: 3.14
```

- A ValueError will occur when attempting to convert a non-numeric string to a float using float()

- Passing any other data-types, like a list, will result in a TypeError:

```
input_value = {'value': 42}
try:
    result = float(input_value)
except TypeError:
    print("Invalid input type for conversion")
```

04/06/2025

NaN

- Floats have a special "Not-a-Number" value `nan`. Typically arises from operations that do not have a meaningful result.

```
result = float('inf') - float('inf')
print(result) # nan
```

'inf' is infinity

Subtracting +ve infinity from +ve infinity, results in `nan` value

```
nan_string = "NaN"
nan_float = float(nan_string)

print(nan_float) # output: nan
```

see how "NaN" is handled

Coercing values to Strings

- The `str()` function used to convert values to strings. Works with all built-in Python data types plus many non-built-in types.

```
str(True) # 'True'
```

- `print()` function automatically converts the value to a string using the `str()` function

```
number = 42
print("The answer is:", number) # output: The answer is: 42
```

04/06/2025

- Even though `print()` coerces its arguments to strings, it does not return the resulting strings, just outputs them. For this reason `print()` is not considered to be coercion.

- String interpolation is used for including values within a string. Python automatically coerces the values to a string when you implement string interpolation, using the `str()` function

```
name = "Karl"
age = 30
message = f"Hello, my name is {name} and I am {age} years old."
print(message) # Output: Hello, my name is Karl and I am 30
                                                    years old.
```

Coercing values to Bodeans

05/06/2025

- The `bool()` is used to convert values to bodeans. Works with all built-in Python values and most non-built-in values
- `bool()` returns `True` if the value is truthy and `False` if the value is falsy.

```
truthy_value = "Hello"
falsy_value = None

is_truthy = bool(truthy_value)
is_falsy = bool(falsy_value)

print(is_trythy) # Output: True
print(is_falsy)  # Output: False
```

converted to `True` as non-empty string

converted to `False` as value is `None`

05/06/2025

repr() vs. str()

- `repr()` function returns a string representation of an object. This object can be used to create a new object.

```
x = 42
repr_x = repr(x) # '42'
new_x = eval(repr_x) # new_x is 42 again
```

NB: CHATGPT PROVIDED EXAMPLE

```
x = [1,2,3]
repr_x = repr(x)
print(repr_x) # Output: [1,2,3]
```

- `str()` function used to convert objects to strings that are ment to be human-readable. Often used for display purposes such as `print()` and string interpolation.
- `repr()` often used for debugging and development purposes. The `repr()` representation should ideally be a valid Python expression that evaluates to original object.
  (See CHATGPT example above)

```
import datetime

today = datetime.datetime.now()
print(str(today)) # 2025-06-05 05:47:05.535262
print(repr(today)) # datetime.datetime(2025, 6, 5, 5, 47, 5, 535262)
```

`str()` makes it easier to read

developer-friendly representation to be used in Python to recreate the same `datetime` object.

05/06/2025

IMPLICIT TYPE COERCION

- The automatic transformation of one data type into another without the programmer's direct instruction.
- Typically occurs during calculations or when mixing distinct data types.

Combining Integer with Float

- When a calculation is performed with both an integer and a float involved, the integer will automatically be adjusted to a float.

```
X = 3       # Integer
y = 2.0     # Float

result = x + y
print(result) # Outputs: 5.0
print(type(result)) # Outputs: <class 'float'>
```

Note the output is a float as the two inputs, x and y, are an integer and float, respectively.

Combining String with Non-string

- Most of the time you cannot coerce non-strings to strings.
  e.g., string + number → TypeError

```
name = "Clare"
age = 35

print(name+age)
# TypeError: can only concatenate str (not "int") to str
```

05/06/2025

- `print()` function is one place where non-strings are automatically converted to strings.

```
name = "Clare"
age = 35

print(name, age) # Outputs: Clare 35
```

As noted earlier, this is not coercion in Python terms. `print()` does not change the variables type - it just temporarily calls `str()` to display the value assigned to the variable