Conditional expressions

- `EXPRESSION` runs only if `CONDITION` is `True`. (Here `UPPERCASE` is a placeholder for code.)

```
if CONDITION:
    EXPRESSION
```

```
x = -3 # example: set x to |x|
if x < 0:
    x = -x

x
```

- One of `True_EXPRESSION` and `False_EXPRESSION` runs:

```
if CONDITION:
    True_EXPRESSION
else:
    False_EXPRESSION
```

```
x = 3 # example
if (x % 2) == 0:
    parity = 'even'
else:
    parity = 'odd'

parity
```

- The first `True` `CONDITION`'s `EXPRESSION` runs; or, if none is `True`, `DEFAULT_EXPRESSION` runs:

```
if CONDITION_1:
    EXPRESSION_1
elif CONDITION_2:      # optional elif ('else if') clauses
    EXPRESSION_2
...
else:                  # optional 'else' clause
    DEFAULT_EXPRESSION # note: first two forms are special cases of this form

def water_state(temperature): # example within a function
    if temperature < 32:
        state = 'frozen'
    elif temperature > 212:
        state = 'boiling'
    else:
        state = 'liquid'
    return state

water_state(20)
water_state(60)
water_state(220)
```

- None of the three forms above works if `CONDITION` has length greater than one. However, `a = np.where(condition, x, y)` sets `a` to be an array with the same length as `condition` with elements from `x` or `y` depending on `condition`. e.g.

```
x = np.array([1, 2, 3])
parity = np.where((x % 2) == 0, 'even', 'odd')
parity
```

More examples

```
def is_heads(): # return True or False to simulate a coin flip
    rng = np.random.default_rng() # no seed here!
    r = rng.normal(size=1)
    if (r < 0):
        return True
    else:
        return False

N = 100 # test is_heads():
coins = np.empty(N)
for i in range(N):
    coins[i] = is_heads()

np.mean(coins)

def is_heads(): # shorter version
    rng = np.random.default_rng()
    r = rng.normal(size=1)
    return r < 0

def is_heads(): # even shorter
    rng = np.random.default_rng()
    return rng.normal(size=1) < 0

def is_heads(): # or, better from the start
    rng = np.random.default_rng()
    return rng.choice([True, False])

# ---------------------------------------------------------------------
def baby_max(a, b): # implements part of np.max(); fails for array
    if a > b:
        return a
    else:
        return b

assert (np.isclose(4, baby_max(3,  4)) & np.isclose(3, baby_max(3, -4)))
x = np.array([3, 3]); y = np.array([4, -4]); m = baby_max(x, y) # BUG

def baby_max(a, b): # improved to work for array
    return np.where(a > b, a, b)

assert (np.isclose(4, baby_max(3,  4)) & np.isclose(3, baby_max(3, -4)))
x = np.array([3, 3]); y = np.array([4, -4]); m = baby_max(x, y)
assert (np.isclose(4, m[0]) & np.isclose(3, m[1]))
```