15-110 Principles of Computing – S21

Lecture 7:
Boolean Types, Conditionals

Teacher:
Gianni A. Di Caro

Boolean type

- `bool`: **Boolean (logical) values**
  - Instances of literals of type `bool` are: `True`, `False`
  - `x = True` defines a Boolean variable with a true value
  - `print(x)` → `True`

```
is_cold = False

lecture_time = True
```

- A **Boolean expression** is an expression that *evaluates* to a Boolean value, true or false
  - `(2+3) > 4` → `True`
  - `5 < x` → `True` or `False` depending on `x`
  - `2*x > y` → `True` or `False` depending on `x, y`

```
x = 2
y = 3
z = (x + y) > 4

a = x > 5

b = a
```

Comparison (Relational) operators

- A Boolean expression results from the application of **comparison operators**

```
a = 3
b = 5
```

○ `x == y`	*x is equal to y*	`c = (a == b)`
○ `x != y`	*x is not equal to y*	`c = (a != b)`
○ `x > y`	*x is greater than y*	`c = (a > b)`
○ `x < y`	*x is less than y*	`c = (a < b)`
○ `x >= y`	*x is greater than or equal to y*	`c = (a >= b)`
○ `x <= y`	*x is less than or equal to y*	`c = (a <= b)`

```
a = True
b = False
c = a > b
    ?
```

`x,y` can be numbers, strings, Boolean, ...

Boolean types and Logical operators: and

- `and`: `(x and y)` evaluates to `True` if and only if both `x` and `y` are `True` expressions
  - `a = ( (2 != 3) and (4.5 > 4)) → True`
  - `a = ( (2 != 2) and (8 == 8)) → False`
  - `a = ( (2 != 3) and (True == False)) → False`
  - `a = ( (2 != 2) and (3 >= 1.5)) →  False`

Logical truth table for AND

p	q	p∧q
T	T	T
T	F	F
F	T	F
F	F	F

- 0 for `False`
- 1 for `True`

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

AND

and: examples of typical applications

- Check whether a value x belongs or not to a certain interval

  is $0 \leq x \leq 5$?

```
in_range = (x >= 0) and (x <=5)
```

- Guarantee that two conditions are *both* satisfied

  is battery more than 95% and the phone is on?

```
conditions_ok = (battery >= 0.95) and (phone == “on”)
```

Boolean types and Logical operators: `or`

- `or`: `(x or y)` evaluates to `True` if and only if either `x` or `y`, or both, are `True` expressions
  - `a = ( (2 != 3) or (4.5 > 4)) → True`
  - `a = ( (2 != 2) or (8 == 8)) → True`
  - `a = ( (2 != 3) or (True == False)) → True`
  - `a = ( (2 != 2) or (3 <= 1.5)) → False`

Logical truth table for OR

p	q	pvq
T	T	T
T	F	T
F	T	T
F	F	F

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

OR

`or`: examples of typical applications

- is color either blue or red?
- is the remainder of $x \div 5$. either 2 or 3?

```
x = x % 3
y =(x == 2) or (x == 5)
```

- x equal to 5, 6, or 7?

```
(x == 5) or (x == 6) or (x == 7)
```

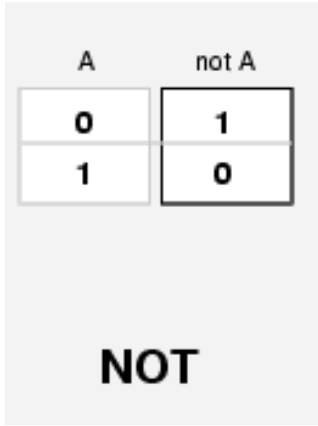
Boolean types and Logical operators: `not`

- `not: (not x)` evaluates to `True` if and only if `x` is a `False` expression
  - `a = not (4.5 < 4)` → `True`
  - `a = not (8 != 7)` → `False`
  - `a = not (False)` → `True`

- Useful anytime the *negation* of an expression is needed

Logical truth table for NOT

p	~p
T	F
F	T



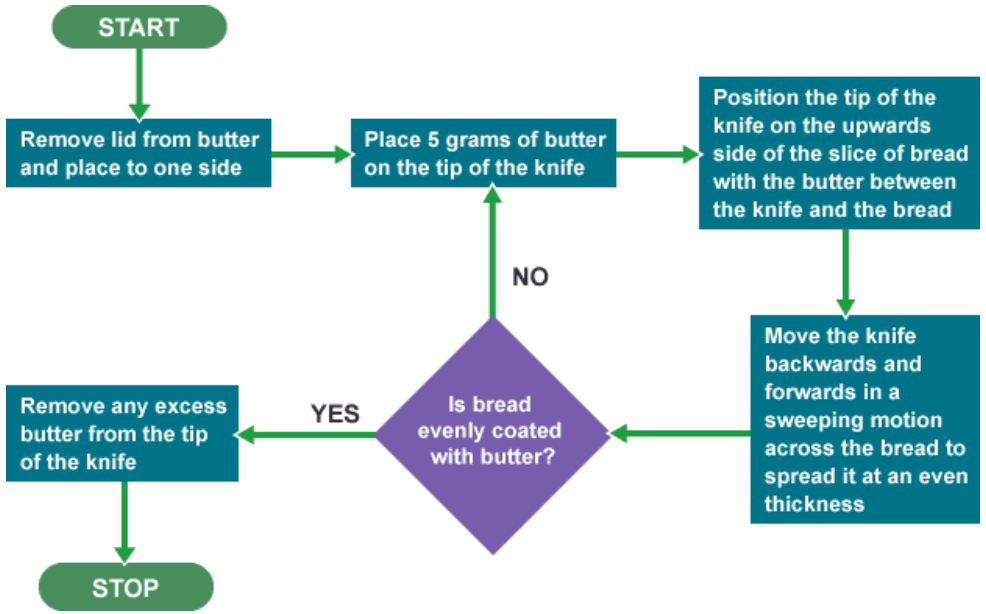
Precedence rules among operators

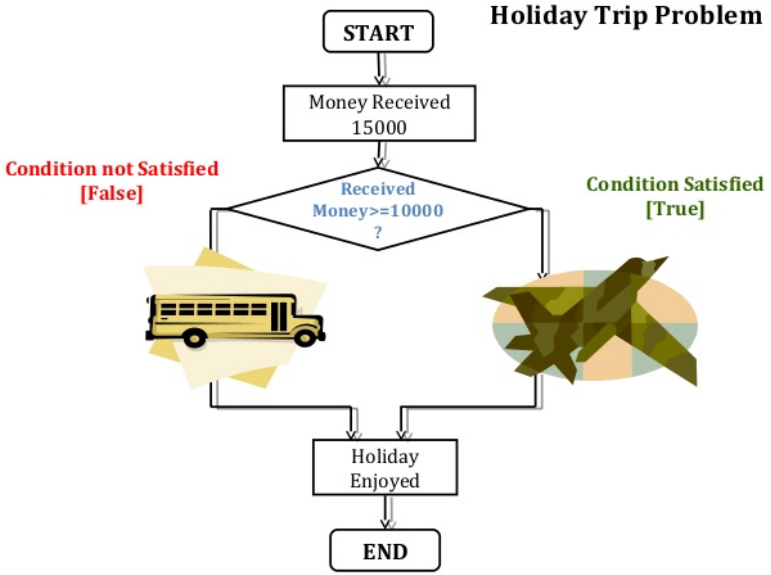
Level	Category	Operators
7(high)	exponent	**
6	multiplication	*,/,//,%
5	addition	+,-
4	relational	==,!=,<=,>=,>,<
3	logical	not
2	logical	and
1(low)	logical	or

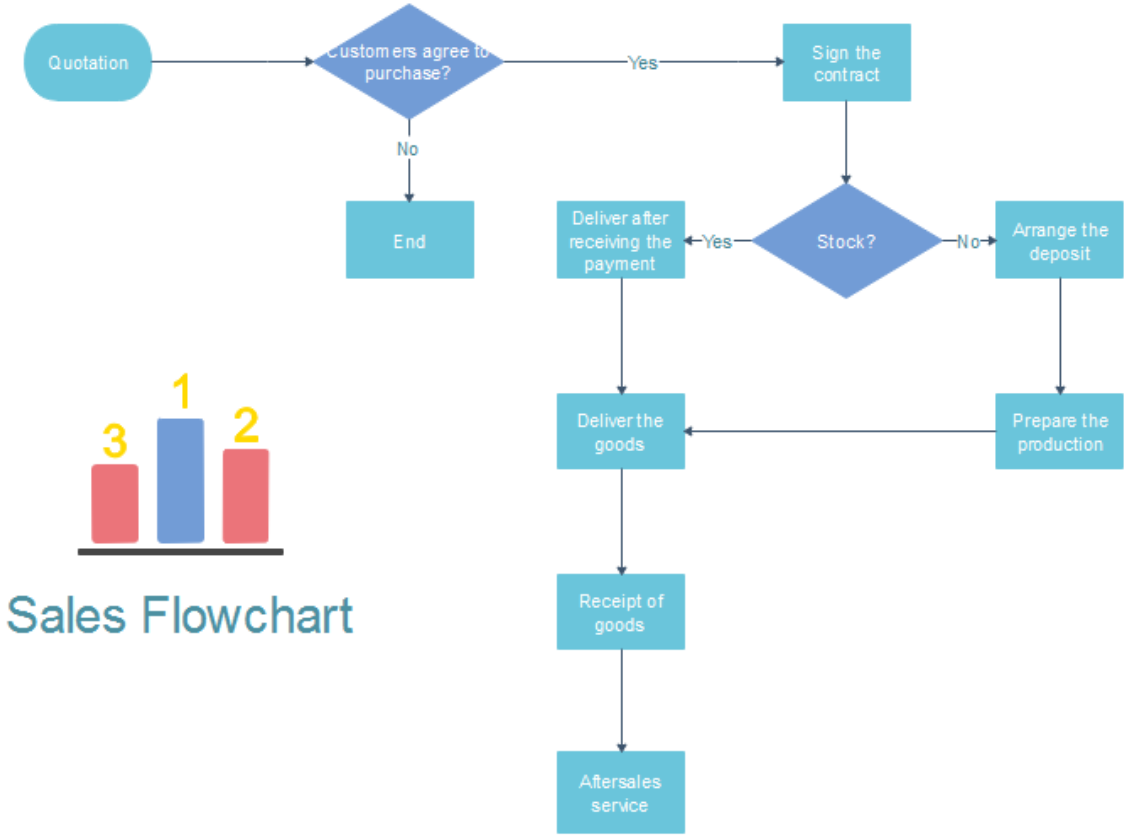
```
x*5 >= 10 and y-6 <= 20
```

First the arithmetic (`x*5`) and then (`y-6`), then the relations (`>= 10`, `<= 20`), and finally the logical `and`

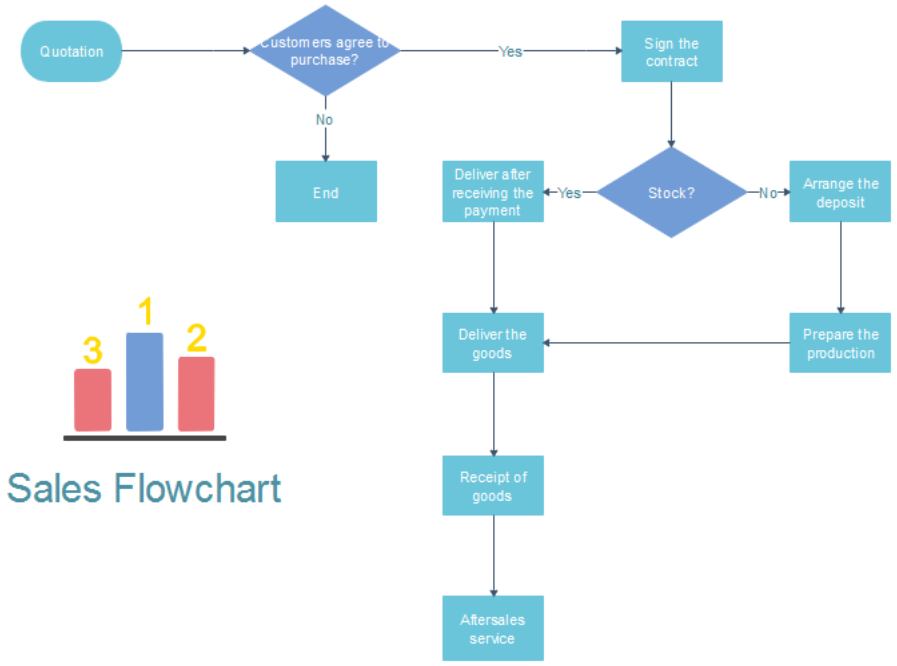
Flow control with conditional execution (branching)



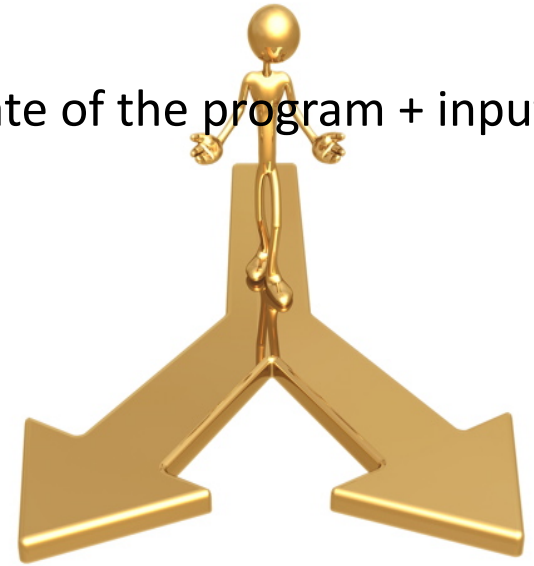




Flow control with conditional execution (branching)

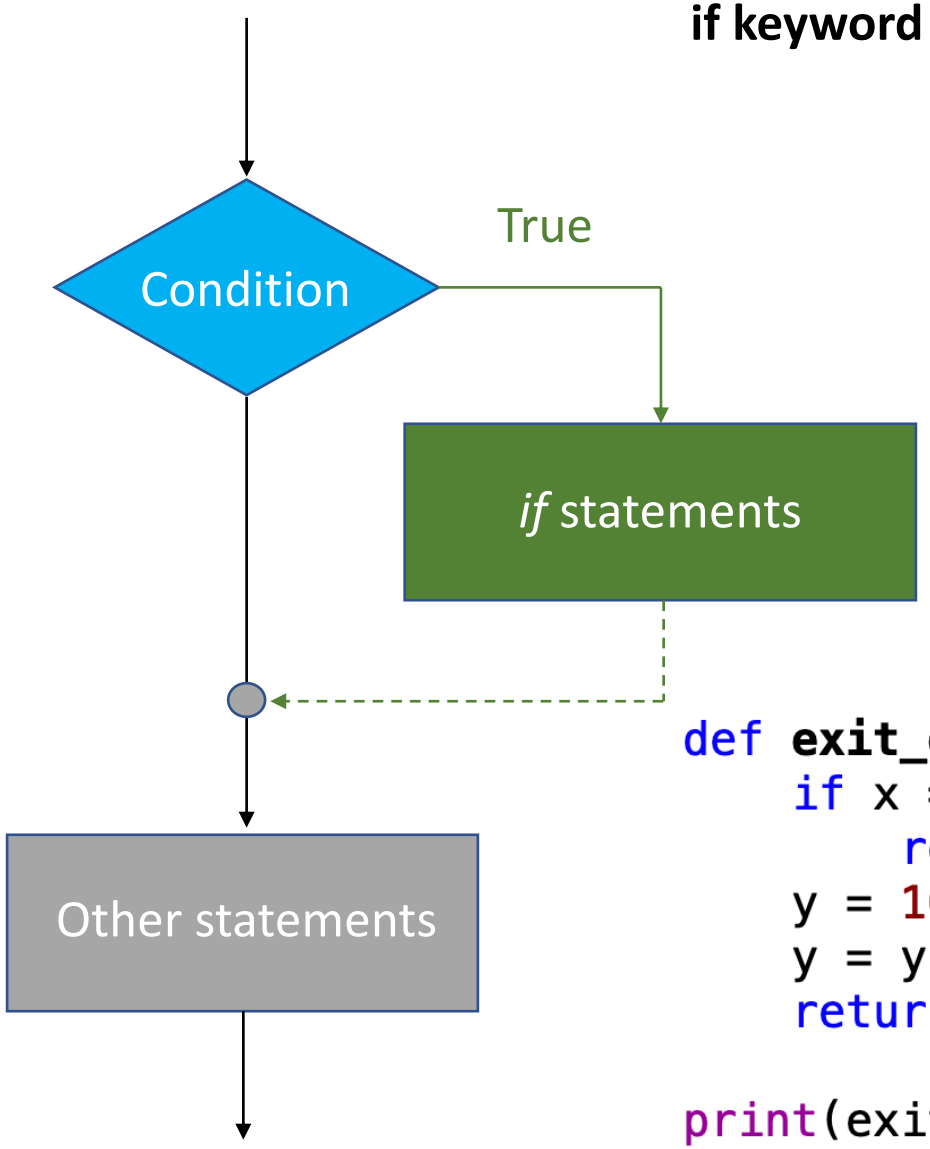


State of the program + input data



Branching …

Flow control with conditional execution: `if`



if keyword

```
if condition_is_true:
    do_something
keep_going_on_with_program
```

Colon

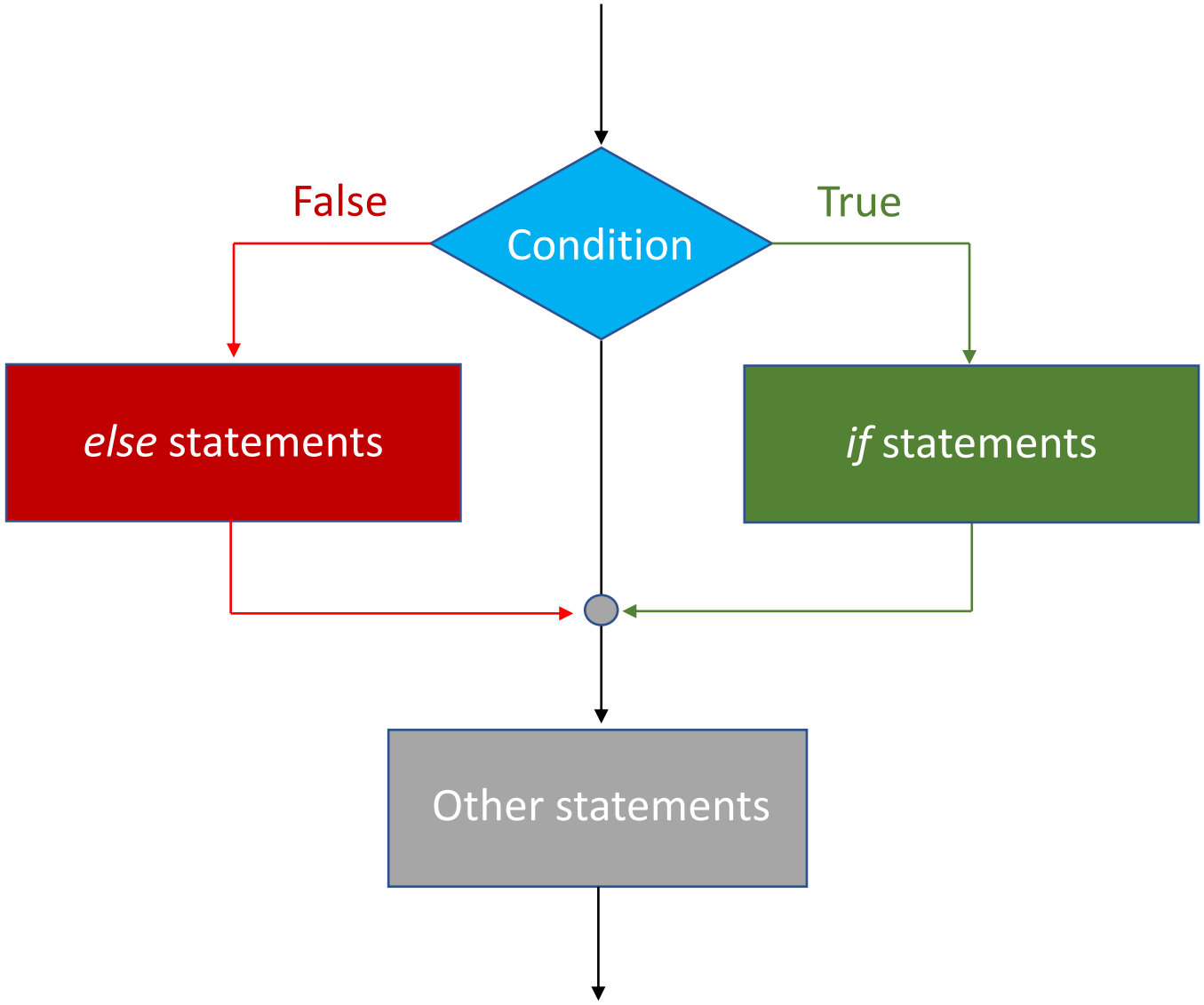
Scope indentation
Use TABS!

```
def exit_on_condition(x):
    if x == True:
        return 1
    y = 10
    y = y ** 2
    return y

print(exit_on_condition(True))
```

```
x = 5
y = 3
if x > 4:
    x = x + 1
y = x
print('x:', x, 'y:', y)
```

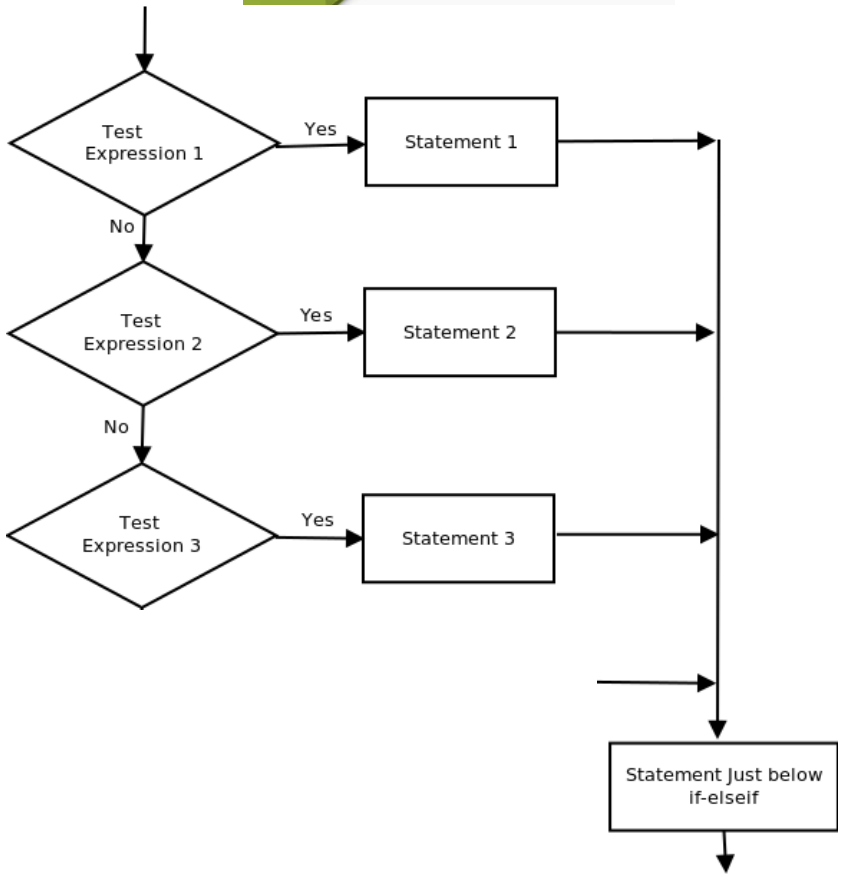
Flow control with conditional execution: `if-else`




```
if condition_is_true:
    do_something
else:
    do_something_else
keep_going_on_with_program
```

```
x = 2
y = 3
if x > 4:
    x = x + 1
else:
    x = 4
y = x
print('x:', x, 'y:', y)
```

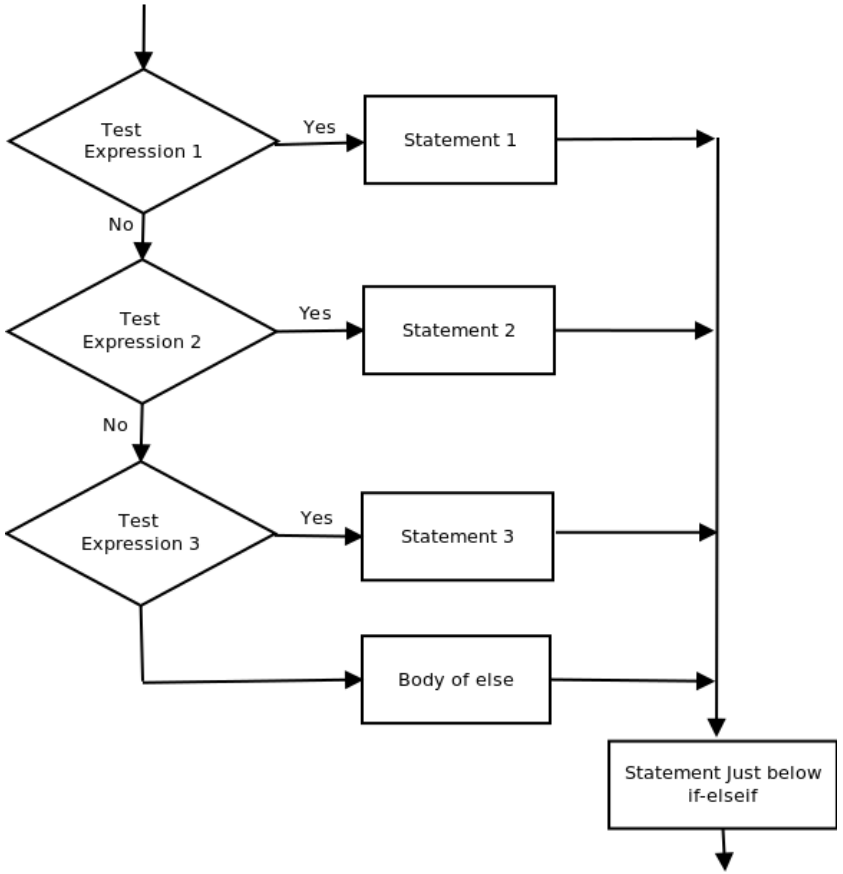
Flow control with conditional execution: `if-elif`




```
if condition_1_is_true:
    do_something_1
elif condition_2_is_true:
    do_something_2
elif condition_3_is_true:
    do_something_3
...
keep_going_on_with_program
```

```
x = 2
y = 3
if x > 4:
    x = x + 1
elif x > 3 and x < 3.5:
    x = x * 2
elif x <= 2:
    x = 1
y = 2 * x
print('x:', x, 'y:', y)
```

Flow control with conditional execution: `if-elif-else`




```
if condition_1_is_true:
    do_something_1
elif condition_2_is_true:
    do_something_2
elif condition_3_is_true:
    do_something_3
...
else:
    do_something_else
keep_going_on_with program
```

```
x = 2
y = 3
if x > 4:
    x = x + 1
elif x > 3 and x < 3.5:
    x = x * 2
elif x <= 2:
    x = 1
else:
    x = 0
y = 2 * x
print('x:', x, 'y:', y)
```

Even or odd?

Write the function `is_odd(n)` that takes as input a number `n` and returns `True` if the number is odd, `False` otherwise

```
def is_odd(n):
    if n % 2 != 0:
        return True
    else:
        return False
```

Max of two numbers

Implement the function `max_two(a, b)` that takes two numbers as input, and returns the largest between them. However, if the two numbers are the same, the function must return -1.

You must write two different implementations of the function:

- Not using `elif`
- Using `elif`

```
def max_two(a,b):
    if a == b:
        return -1
    if a > b:
        return a
    else:
        return b
```

```
def max_two(a,b):
    if a == b:
        return -1
    elif a > b:
        return a
    else:
        return b
```

Weather classification

Write the function `comfortable_weather(t)` that takes as input a number `t` representing current temperature and returns:

- 1 if the temperature is between 14 and 27 (both included),
- 0 if the temperature is higher than 27 and lower than 32 (both included), or lower than 14 and higher than 8 (both included)
- -1 if the temperature is higher than 32 or lower than 8

```
def comfortable_weather(t):
    if (t >= 14) and (t <= 27):
        return 1
    elif (t > 27 and t <= 32) or (t < 14 and t >= 8):
        return 0
    else:
        return -1
```

Max of three numbers

Implement the function `max_three(a, b, c)` that takes three numbers as input, and returns the largest between them.

```
def max_three(a,b,c):
    if a >= b and a >= c:
        return a
    if b >= a and b >= c:
        return b
    if c >= a and c >= b:
        return c
```