Control structures

- input()
- if-elif-else
- while-break-continue

input

- The builtin function `input(`*message*`)` prints *message*, and waits for the user provides a line of input and presses return. The line of input is returned as a `str`
- If you e.g. expect input to be an `int`, then remember to convert the input using `int()`

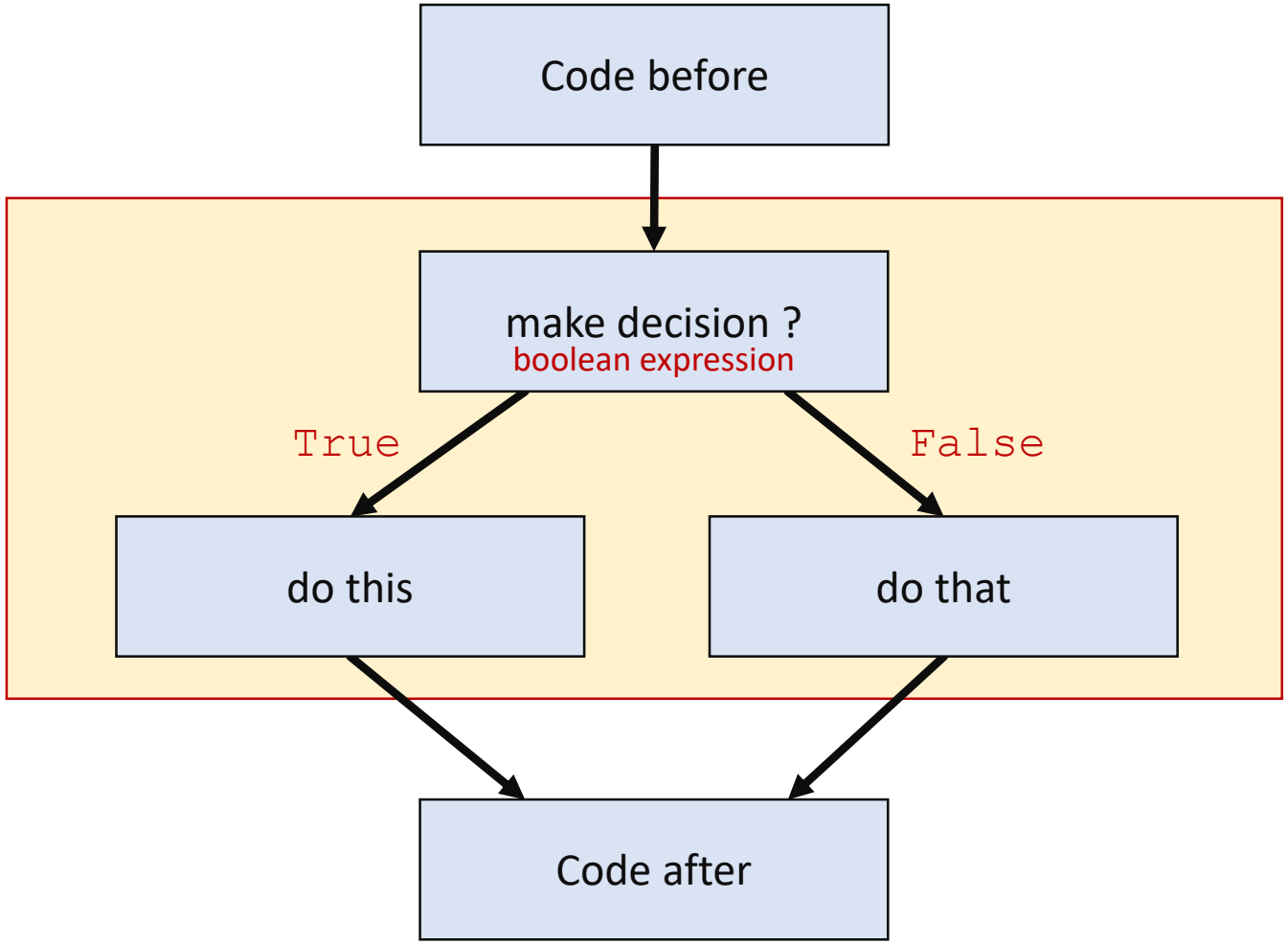
name_age.py

```
name = input("Name: ")
age = int(input("Age: "))
print(name, "is", age, "years old")
```

Output

```
Name: Donald Duck
Age: 84
Donald Duck is 84 years old
```

Branching – do either this or that ?



Basic if-else

```
if boolean expression:
    code
    code
    code
else:
    code
    code
    code
```

identical indentation

identical indentation

if-else.py

```
if x%2 == 0:
    print("even")
else:
    print("odd")
```

Identical indentation for a sequence of lines = the same spaces/tabs should precede code

pass

- `pass` is a Python statement doing nothing. Can be used where a statement is required but you want to skip (e.g. code will be writen later)

- Example (bad example, since `else` could just be omitted):

if-else.py

```
if x%2 == 0:
    print(”even”)
else:
    pass
```

if-elif-else

```
if condition:
        code
elif condition: # zero or more “elfi“ ≡ “else if”
        code
else:  # optional
        code
```

```
if (condition) {
  code
} else if (condition) {
  code
} else {
  code
}
```

Java, C, C++ syntax

Other languages using indentation for blocking:
ABC (1976), occam (1983), Miranda (1985)

if.py

```
if x == 0:
    print("zero")
```

if-else.py

```
if x%2 == 0:
    print("even")
else:
    print("odd")
```

elif.py

```
if x < 0:
    print("negative")
elif x == 0:
    print("zero")
elif x == 1:
    print("one")
else:
    print(">=2")
```

Questions – What value is printed?

```
x = 1
if x == 2:
    x = x + 1
else:
    x = x + 1
    x = x + 1
x = x + 1
print(x)
```

a) 1
b) 2
c) 3
 d) 4
e) 5
f) Don’t know

Nested if-statements

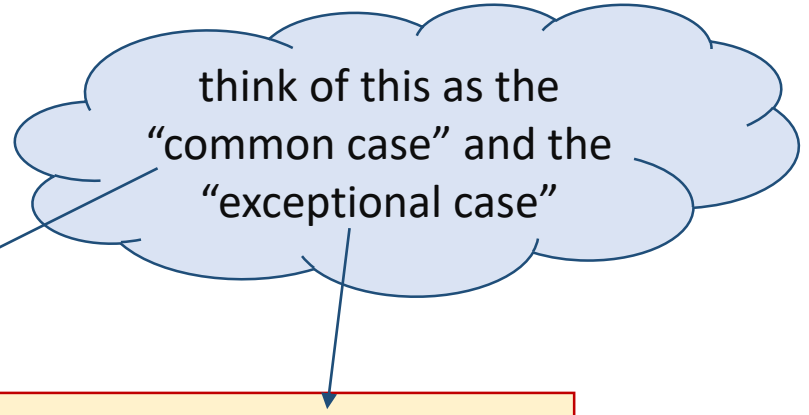
nested-if.py

```
if x < 0:
    print("negative")
elif x%2 == 0:
    if x == 0:
        print("zero")
    elif x == 2:
        print("even prime number")
    else:
        print("even composite number")
else:
    if x == 1:
        print("one")
    else:
        print("some odd number")
```

if-else *expressions*

- A very common computation is

```
if test:
    x = true-expression
else:
    x = false-expression
```



- In Python there is a shorthand for this:

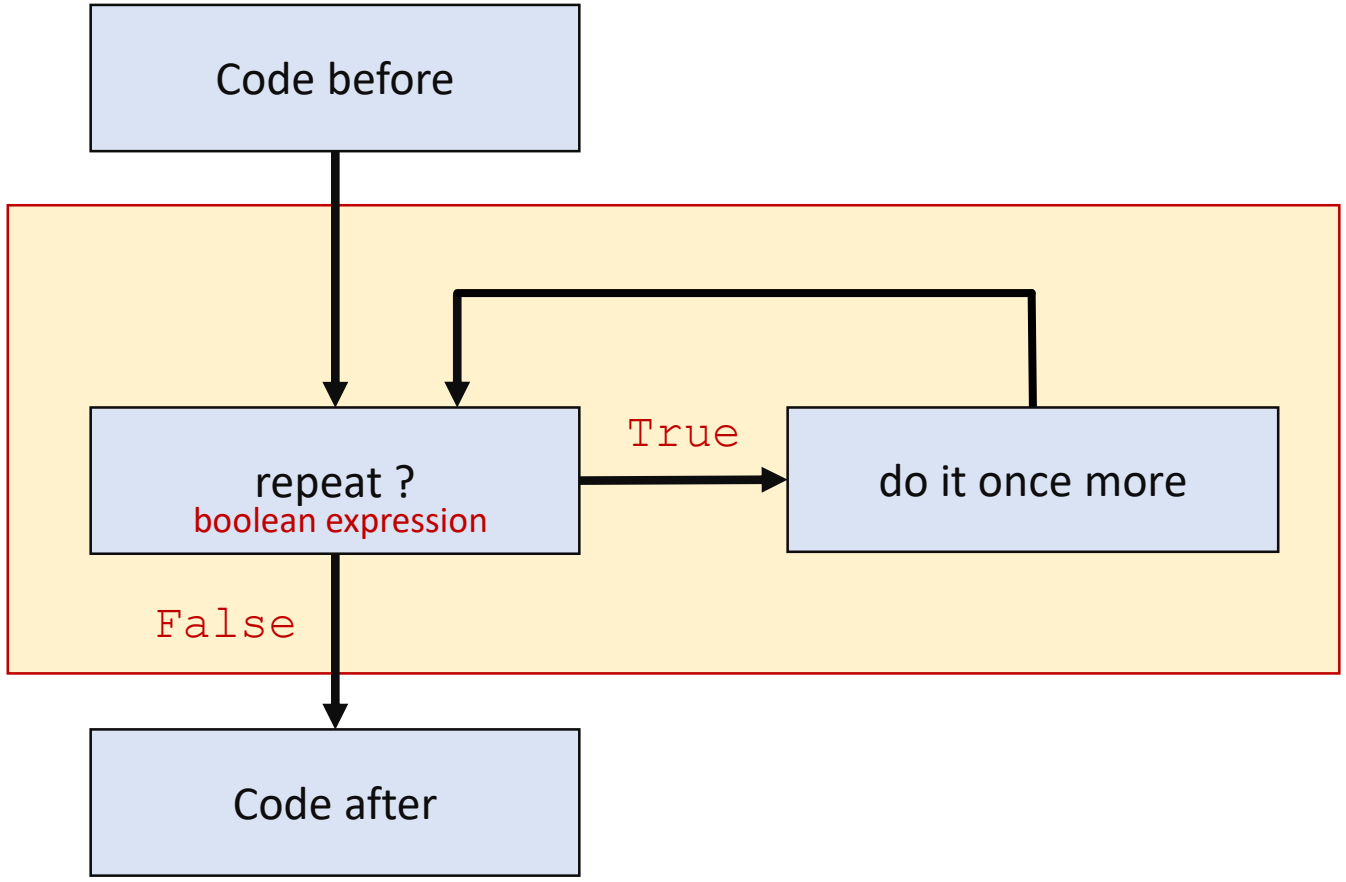
```
x = true-expression if test else false-expression
```

(see What’s New in Python 2.5 - **PEP 308: Conditional Expressions**)

- In C, C++ and Java the equivalent notation is (note the different order)

```
x = test ? true-expression : false-expression
```

Repeat until done



while-statement

```
while condition:
    code
    ...
    break    # jump to code after while loop
    ...
    continue # jump to condition at the
    ...      # beginning of while loop
```

```
while (condition) {
  code
}
```

Java, C, C++ syntax

count.py

```
x = 1
while x <= 10:
    print(x)
    x = x + 1
```

int-sqrt.py

```
low = 0
high = x+1
while True: # low <= sqrt(x) < high
    if low+1 == high:
        break
    mid = (high+low) // 2
    if mid*mid <= x:
        low = mid
        continue
    high = mid
print(low) # low = floor(sqrt(x))
```