

# 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()`

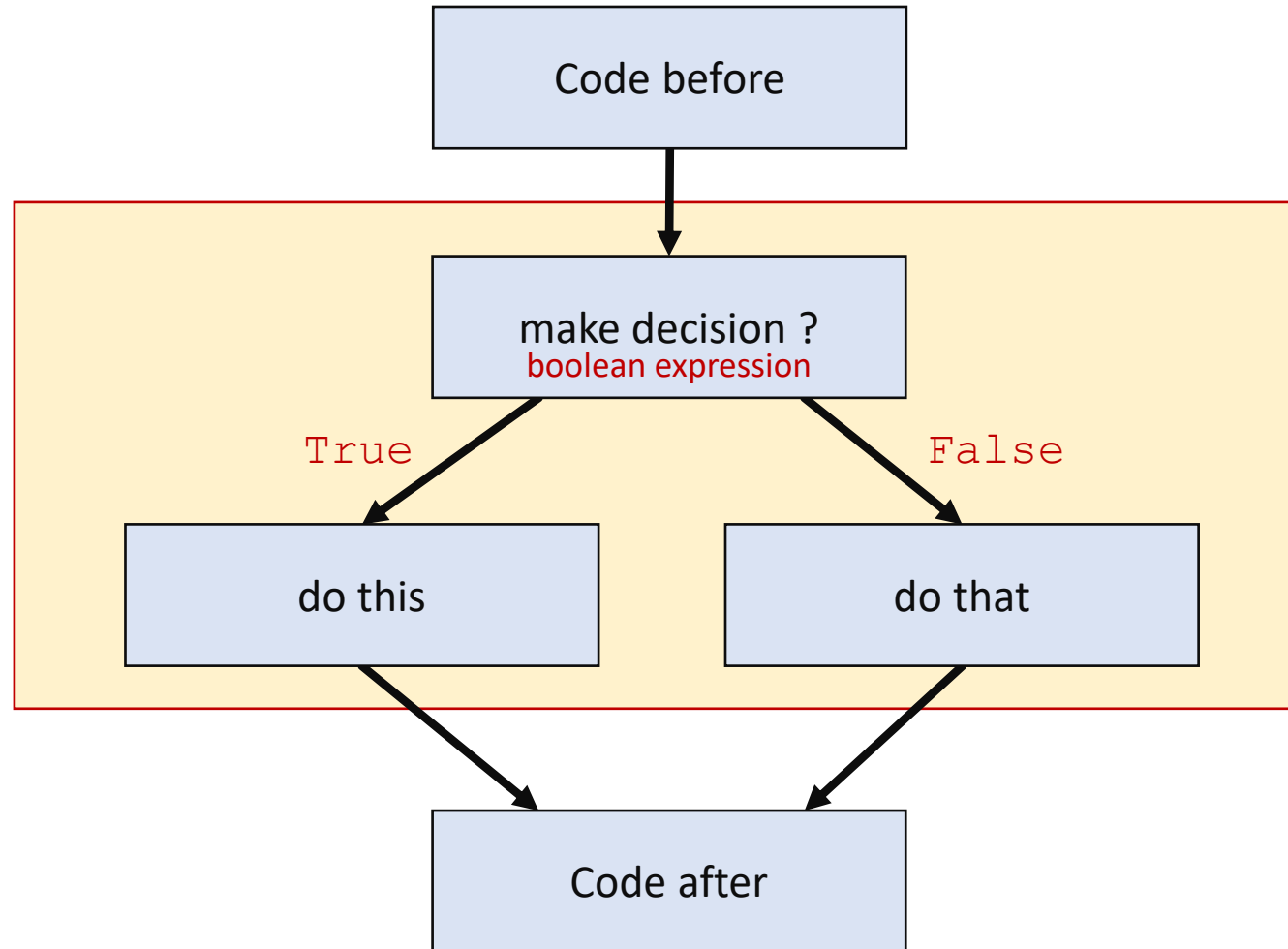
## name-age.py

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

## Python shell

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

# Branching – do either this or that ?



# Basic if-else

`if` *boolean expression*:

identical  
indentation { *code*  
*code*  
*code*

`else:`

identical  
indentation { *code*  
*code*  
*code*

**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 written 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')
```



# Common mistake

**if-if.py**

```
x = int(input())
if x == 0:
    print('zero')
if x % 2 == 0:
    print('even')
```

**Python shell**

```
> 0
| zero
| even
```



**if-elif.py**

```
x = int(input())
if x == 0:
    print('zero')
elif x % 2 == 0:
    print('even')
```

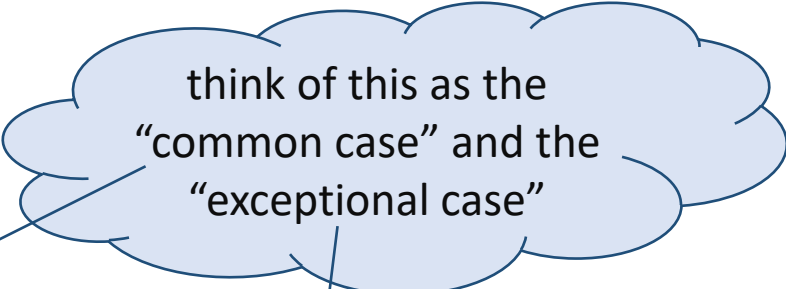
**Python shell**

```
> 0
| zero
```

# if-else expressions

- A very common computation is

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



think of this as the  
“common case” and the  
“exceptional case”

- In Python there is a shorthand for this:

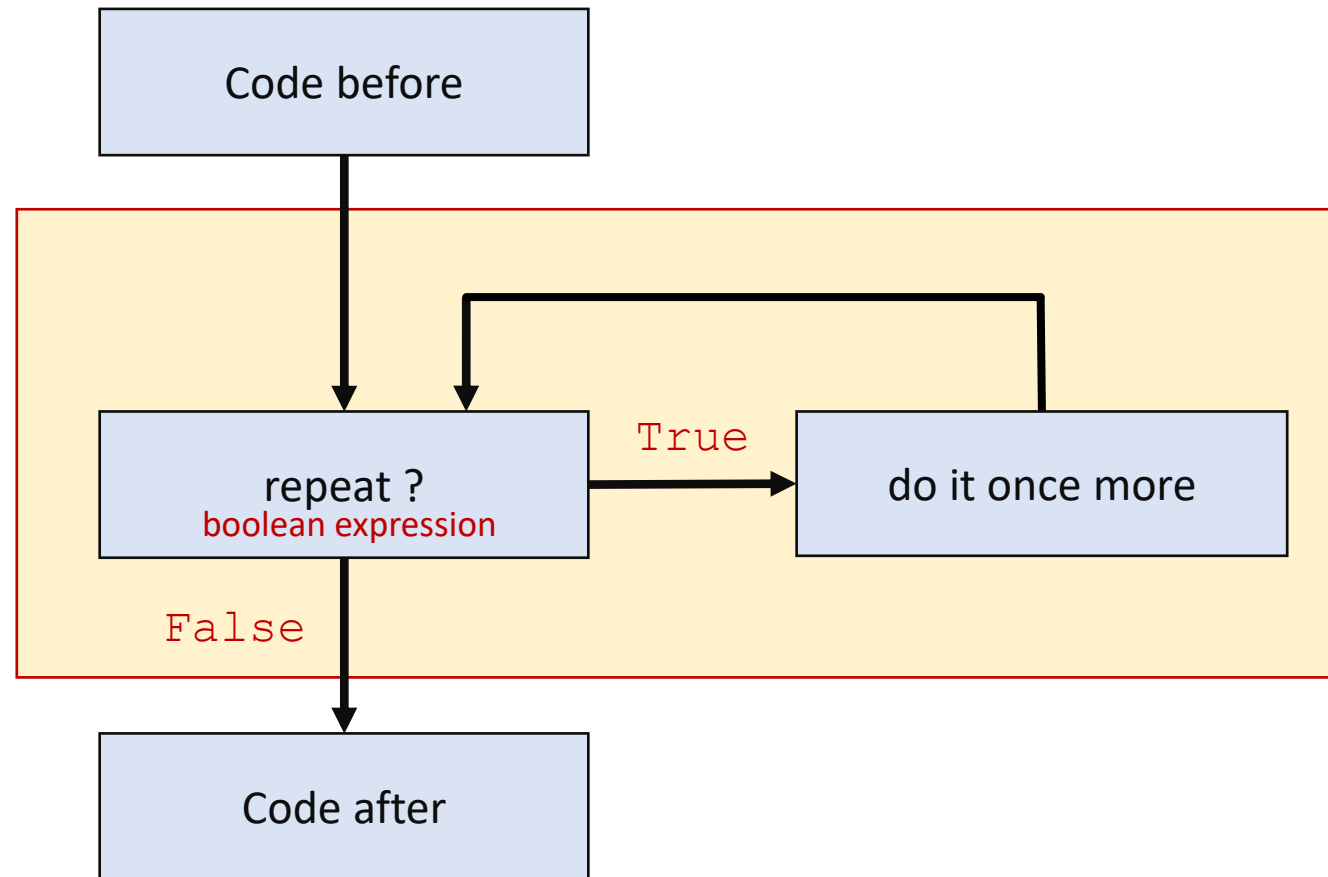
```
x = true-expression if test else false-expression
```

(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 <= 5:  
    print(x, end=' ')  
    x = x + 1  
print('and', x)
```

## Python shell

```
| 1 2 3 4 5 and 6
```

The function `randint(a, b)` from module `random` returns a random integer from  $\{a, a + 1, \dots, b - 1, b\}$

## random-pair.py

```
from random import randint  
while True:  
    x = randint(1, 10)  
    y = randint(1, 10)  
    if abs(x - y) >= 2:  
        break  
    print('too close', x, y)  
print(x, y)
```

## Python shell

```
| too close 4 4  
| too close 10 9  
| 8 5
```

An exercise asks to  
simplify the code

# Computing $\lfloor \sqrt{x} \rfloor$ using binary search

**int-sqrt.py**

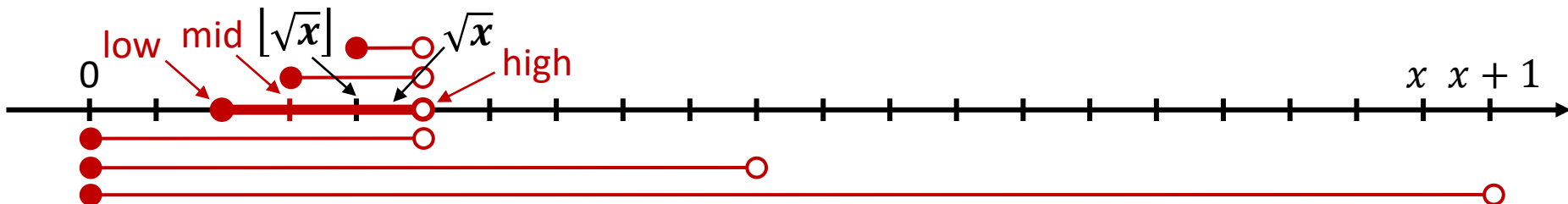
```
x = 20
low = 0
high = x + 1
while True: # low <= sqrt(x) < high
    if low + 1 == high:
        break
    mid = (high + low) // 2
    if mid * mid <= x:
        low = mid
        continue
    high = mid
print(low) # low = floor(sqrt(x))
```

Integer division

$$\left\lfloor \frac{\text{high} + \text{low}}{2} \right\rfloor$$

$$\text{mid} \leq \sqrt{x}$$

$$\Updownarrow$$
$$\text{mid}^2 \leq x$$

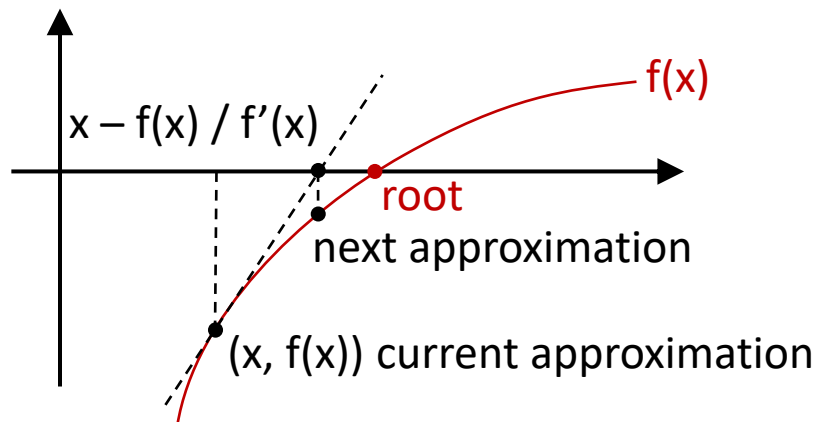


# Division using the Newton-Raphson method

- **Goal:** Compute  $1 / n$  only using +, -, and \*
- $x = 1 / n \iff f(x) = n - 1 / x = 0$
- Problem reduces to finding **root** of  $f$
- Newton-Raphson:

$$x := x - f(x)/f'(x) = x - (n - 1/x)/(1/x^2) = (2 - n \cdot x) \cdot x$$

since  $f'(x) = 1 / x^2$  for  $f(x) = n - 1 / x$



## division.py

```
n = 0.75 # n in [0.5, 1.0]
x = 1.0
last = 0.0
while last < x:
    print(x)
    last = x
    x = (2 - n * x) * x
print('Apx of 1.0 /', n, '=', x)
print('Python 1.0 /', n, '=', 1.0 / n)
```

## Python shell

```
| 1.0
| 1.25
| 1.328125
| 1.33331298828125
| 1.3333333330228925
| 1.3333333333333333
| Apx of 1.0 / 0.75 = 1.3333333333333333
| Python 1.0 / 0.75 = 1.3333333333333333
```