Program flow, variables, conditionals, essential pieces

Tomáš Svoboda, http://cmp.felk.cvut.cz/~svoboda

Programming Essentials, EECS, CTU in Prague

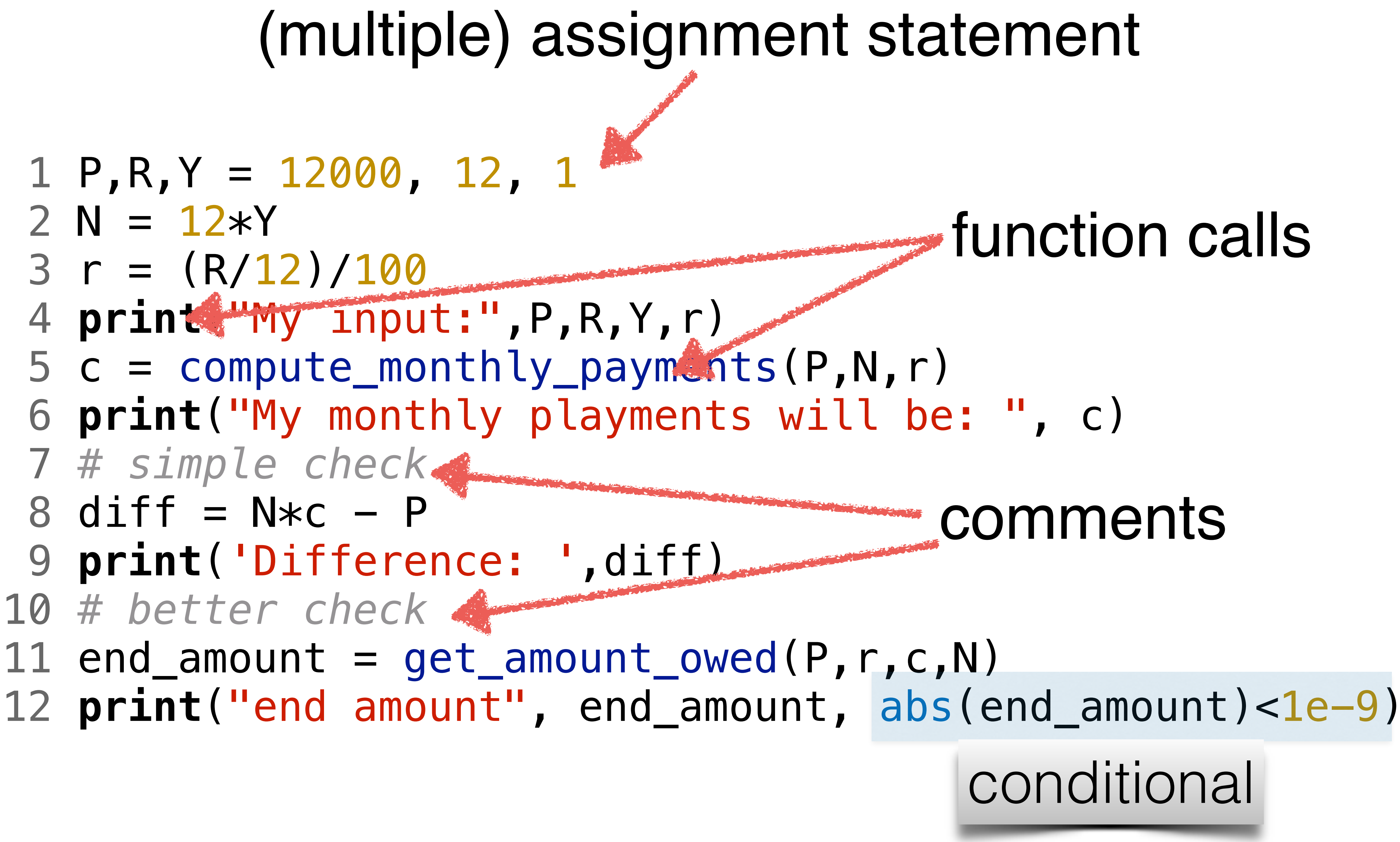
```python
__author__ = 'svoboda'

def compute_monthly_payments(P,N,r):
    c_multiplicator = 1
    for i in range(1,N):
        c_multiplicator = c_multiplicator + (1+r)**i
    return (((1+r)**N)*P) / c_multiplicator

def get_amount_owed(P,r,c,m):
    if m==0:
        return P
    previous_amount = get_amount_owed(P,r,c,m-1)
    return (1+r)*previous_amount - c

P,R,Y = 12000, 12, 1
N = 12*Y
r = (R/12)/100
print("My input:",P,R,Y,r)
c = compute_monthly_payments(P,N,r)
print("My monthly playments will be: ", c)
# simple check
diff = N*c - P
print('Difference: ',diff)
# better check
end_amount = get_amount_owed(P,r,c,N)
print("end amount", end_amount, abs(end_amount)<1e-9)
```

sequence of instructions



(multiple) assignment statement

function calls

comments

```
P,R,Y = 12000, 12, 1
N = 12*Y
r = (R/12)/100
print("My input:",P,R,Y,r)
c = compute_monthly_payments(P,N,r)
print("My monthly playments will be: ", c)
# simple check
diff = N*c - P
print('Difference: ',diff)
# better check
end_amount = get_amount_owed(P,r,c,N)
print("end amount", end_amount, abs(end_amount)<1e-9)
```

conditional

variables

- integers (int), `4,7,8`
- strings (str), “`hello`”
- floats (float), `1.0, 5.7`
- `type(1.0)`

How to name variables

- the longer life the longer name
- the more important the longer name
- think about *readability* of the code
- a meaningfull name does not add the meaning just by itself. The code must do this.

reserved names

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with
yield	True	False	None		

avoid also some too generic

- max, min, abs
- list, string, array
- be specific, descriptive

statement

- an instruction the Python can *execute*
- does not produce any result
- `day = “Saturday”` is a statement
- we will see more …

expressions

- evaluation of an expression produces a value
- `1+1`
- `abs(-3)`
- …

operators and operands

- operand operator operand
- `1 + 3`
- `6/4 vs 6//4` (floor division)
- `7%4` (modulus operator)

order of operations - PEMDAS

1. **P**arentheses

2. **E**xponentiation

3. **M**ultiplication and **D**ivision

4. **A**ddition and **S**ubtraction

left-to-right evaluation on the same level, with the exception of exponentiation (**)

operators and data types

- Python is very flexible in this
- one symbol can have different meaning depending on the data type(s)

converting types

- comfortable, especially strings to numbers and back
- may help
- use wisely

input

- get an input from the user
- the result is a `str` data type
- type conversion

assignment = not like the math =

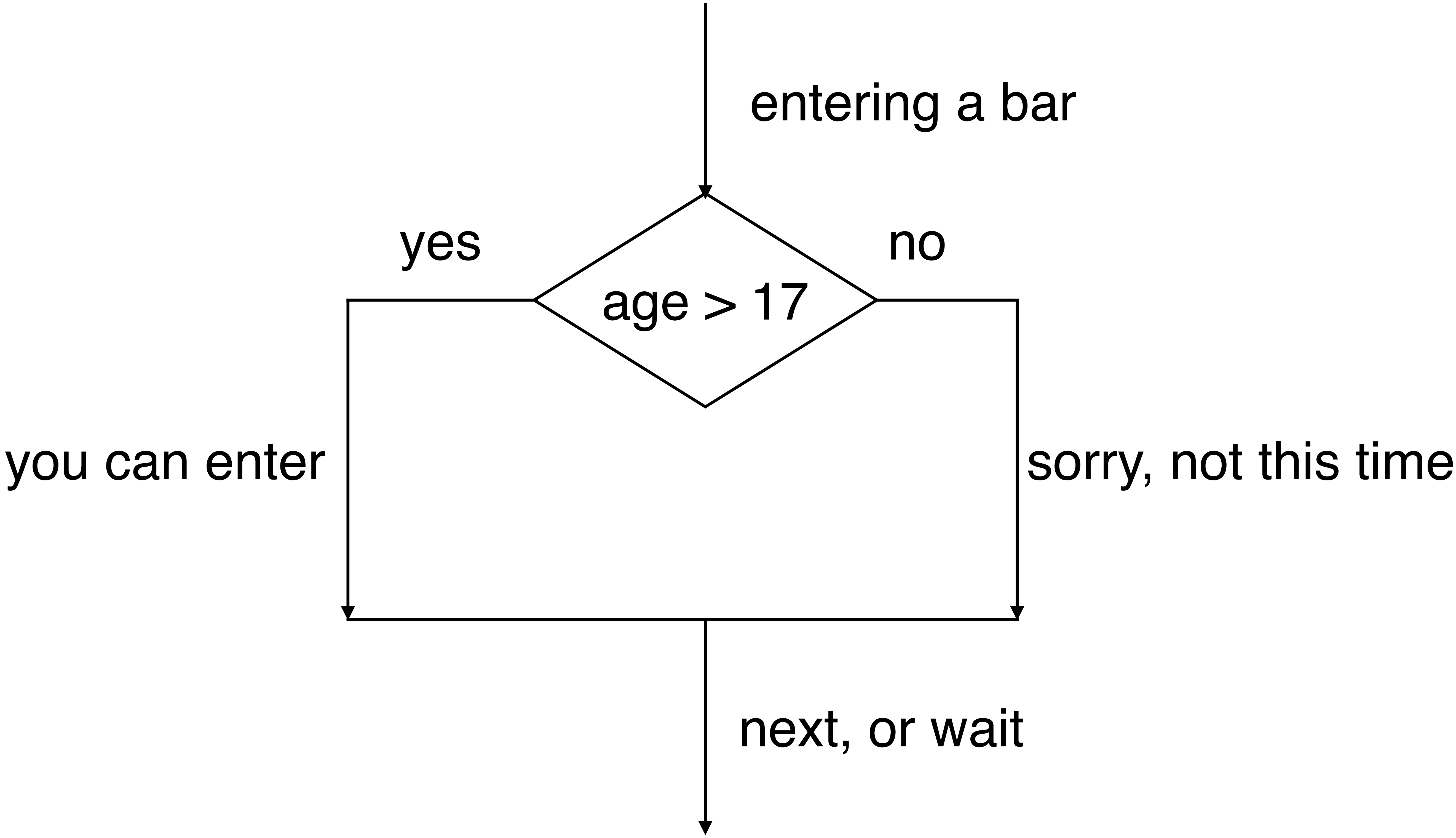
```
a = 4
b = 5
a = a+b
```

- the variables can change over time
- think about score in a game
- what is the difference between a=a+b and a==a+b?

Conditionals

what is it all about

- test some condition
- change the program behaviour accordingly



entering a bar

yes

age > 17

no

you can enter

sorry, not this time

next, or wait

comparison operators

```
x == y                  # Produce True if ... x is equal to y
x != y                  # ... x is not equal to y
x > y                   # ... x is greater than y
x < y                   # ... x is less than y
x >= y                  # ... x is greater than or equal to y
x <= y                  # ... x is less than or equal to y
```

truth tables

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
F	F	F
F	T	T
T	F	T
T	T	T

a	not a
F	T
T	F

simplifying comparisons

- make it simple
- `a and False = ?`
- `a and True = ?`
- `a or True = ?`

logical opposites

operator	logical opposite
==	!=
!=	==
<	>=
<=	>
>	<=
>=	<

```
if not (age >= 17):
    print("Hey, you’re too young to get a driving licence!")

if age < 17:
    print("Hey, you’re too young to get a driving licence!")
```

De Morgan’s laws

```
not (x and y)  ==  (not x) or (not y)
not (x or y)   ==  (not x) and (not y)
```

can you attack the dragon or not?

```
if not ((sword_charge >= 0.90) and (shield_energy >= 100)):
```

and what about this?

```
if (sword_charge < 0.90) or (shield_energy < 100):
```