CONDITIONAL EXECUTION
Comparison operators

<  less than
<=  less than or equal to
==  equal to
>=  greater than or equal to
greater than
!=  not equal

Remember:  = is used for assignment
Logical operators

and
or
not

if x > 0 and x < 10:
    print('x is greater than 0 and less than 10')

if n % 2 == 0 or n % 3 == 0:
    print('n is divisible by 2 or by 3')

if not (n % 2 == 0):
    print('n is odd')
One-way decisions

```python
if question:
    statement
    statement
    statement

if x > 0:
    print('Positive')
    print('Greater than 0')
```
Two-way decisions

```python
if question:
    statement
else:
    statement
```

```python
if x > 0:
    print('Positive')
else:
    print('Negative or zero')
```
Multi-way decisions

```python
if question:
    statement
elif question:
    statement
else:
    statement
```

```python
if x > 0:
   print('Positive')
elif x < 0:
   print('Negative')
else:
   print('Zero')
```
Indentation

• **Increase indent** after an if statement (do not forget about : )

• **Maintain indent** to indicate the scope of the block (which lines are affected by the if)

• **Reduce indent** back to the level of the if statement to indicate the end of the block

• Usually 4 spaces

```python
if x > 0:
    print('Positive')
    print('Greater than 0')
print('The end')
```
Nested decisions

if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
Try-except

try :
    statement
    statement
except :
    statement
    statement

try :
    number = int(string)
except :
    number = -1
Reserved words

if
    and
    or
    not
else
elif
try
except
Each value has type

```python
>>> type(400)
<class 'int'>

>>> type(2.5)
<class 'float'>

>>> type(True)
<class 'bool'>

>>> type('True')
<class 'str'>
```
Pair programming

• One programmer is the driver and writes code. The other is the observer or navigator who reviews each line and plans further work.

• Benefits for the code
  • About 15%-20% fewer bugs
  • Better quality of the code
  • Better design of the code

• Benefits for the programmers
  • Increases knowledge sharing
  • Improves focus
  • Improves communication skills
  • Reduces bad programming habits
Learning through pair programming

• Pairing helps to introduce new ideas and question established practices. More ways to solve the problem will be considered than a single programmer might do.

• What you feel is good for you and what is good for you by objective measures are often two different things.

• You truly understand something only when you are able to explain it to your grandmother. If your partner doesn't understand it, it's your fault!