bool types and if statements

bool types and if statements

Basic program control flow in Python

List of Basic Datatypes

We have already seen three of the basic data types in Python:

- Integers (`int type`)
  - $\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$
- Floating-point (decimal/fractional) numbers (`float type`)
  - $1.5, -1.5, 1.0, ...$
- Strings (words and sentences) (`str type`)
  - `"Tyler"`, `"hello world"`, `"%#()*"`, `"1.0"`

List of Basic Datatypes

We have already seen three of the basic data types in Python:

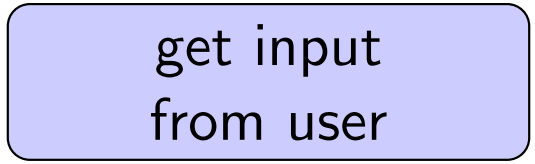
- Integers (`int type`)
  - $\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$
- Floating-point (decimal/fractional) numbers (`float type`)
  - $1.5, -1.5, 1.0, ...$
- Strings (words and sentences) (`str type`)
  - `"Tyler"`, `"hello world"`, `"%#()*"`, `"1.0"`

One more type in the future:

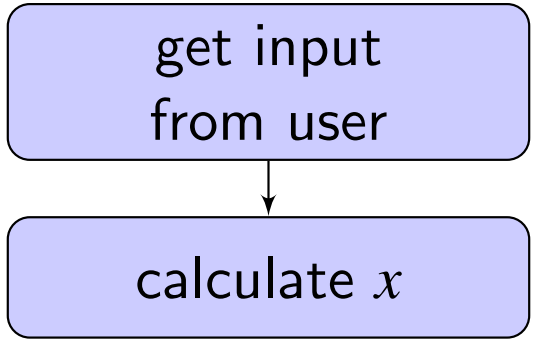
- **Boolean (`bool type`)**
  - `True`, `False`

Linear Control Flow

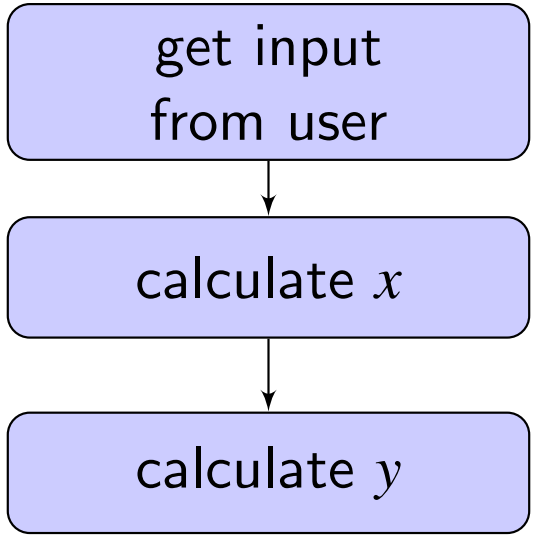
Linear Control Flow



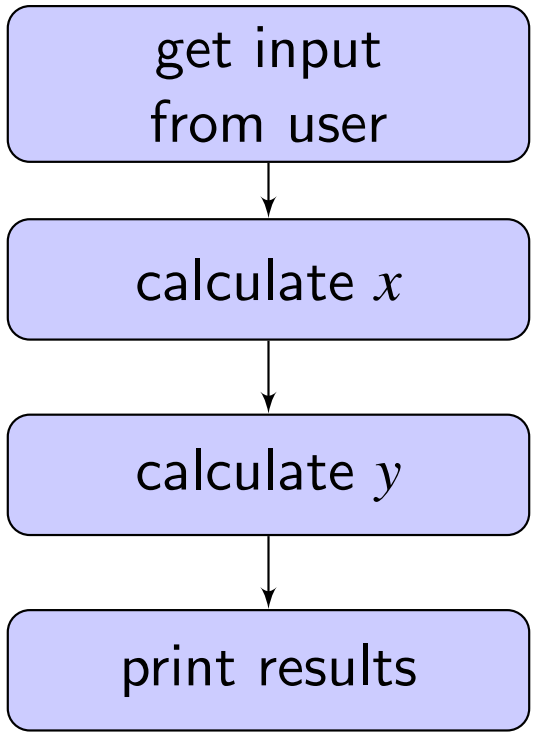
Linear Control Flow



Linear Control Flow



Linear Control Flow

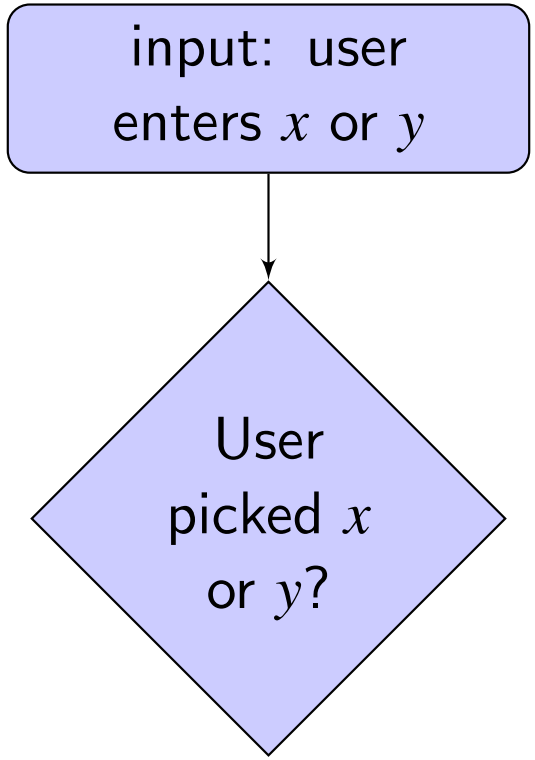


Branching Control Flow

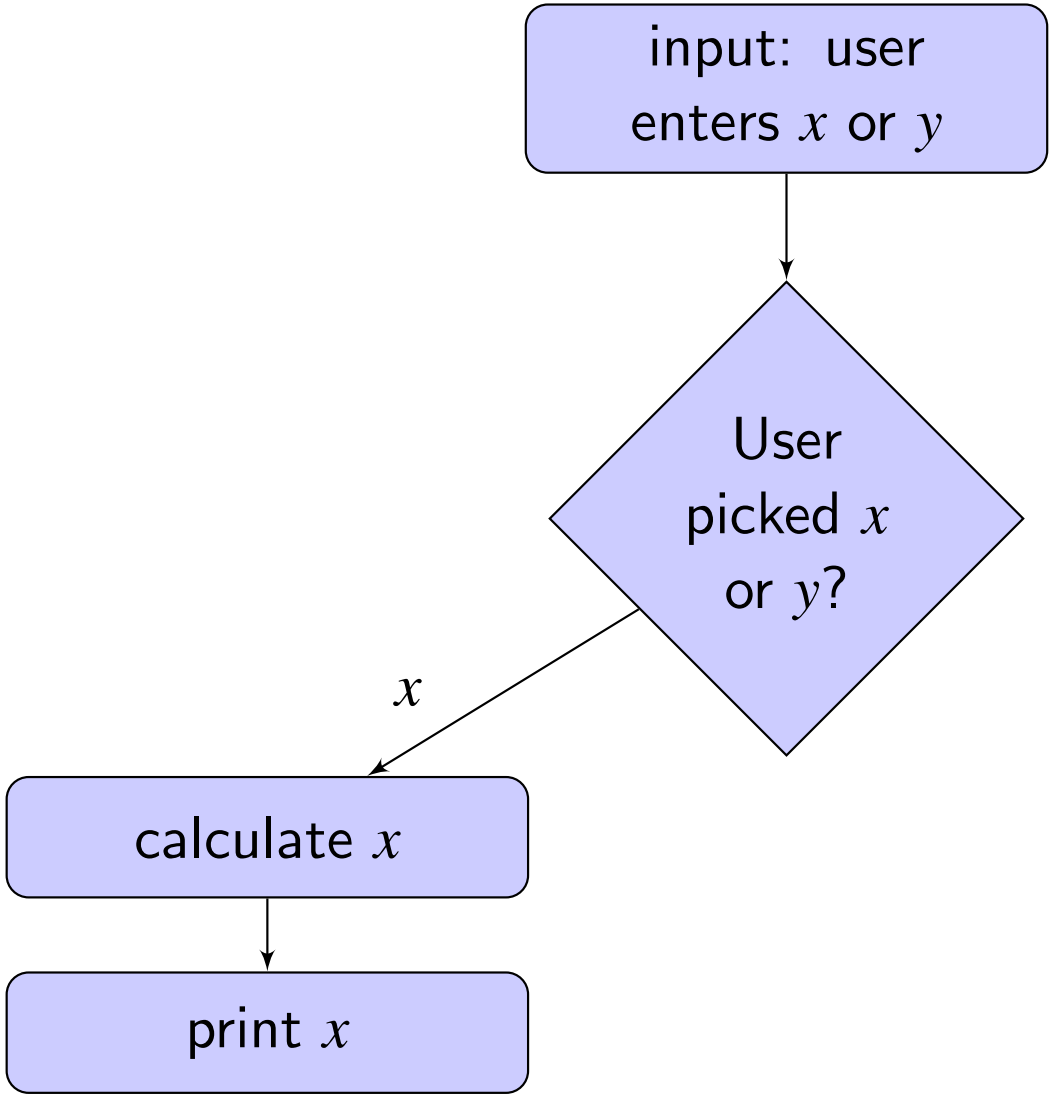
Branching Control Flow

input: user enters x or y

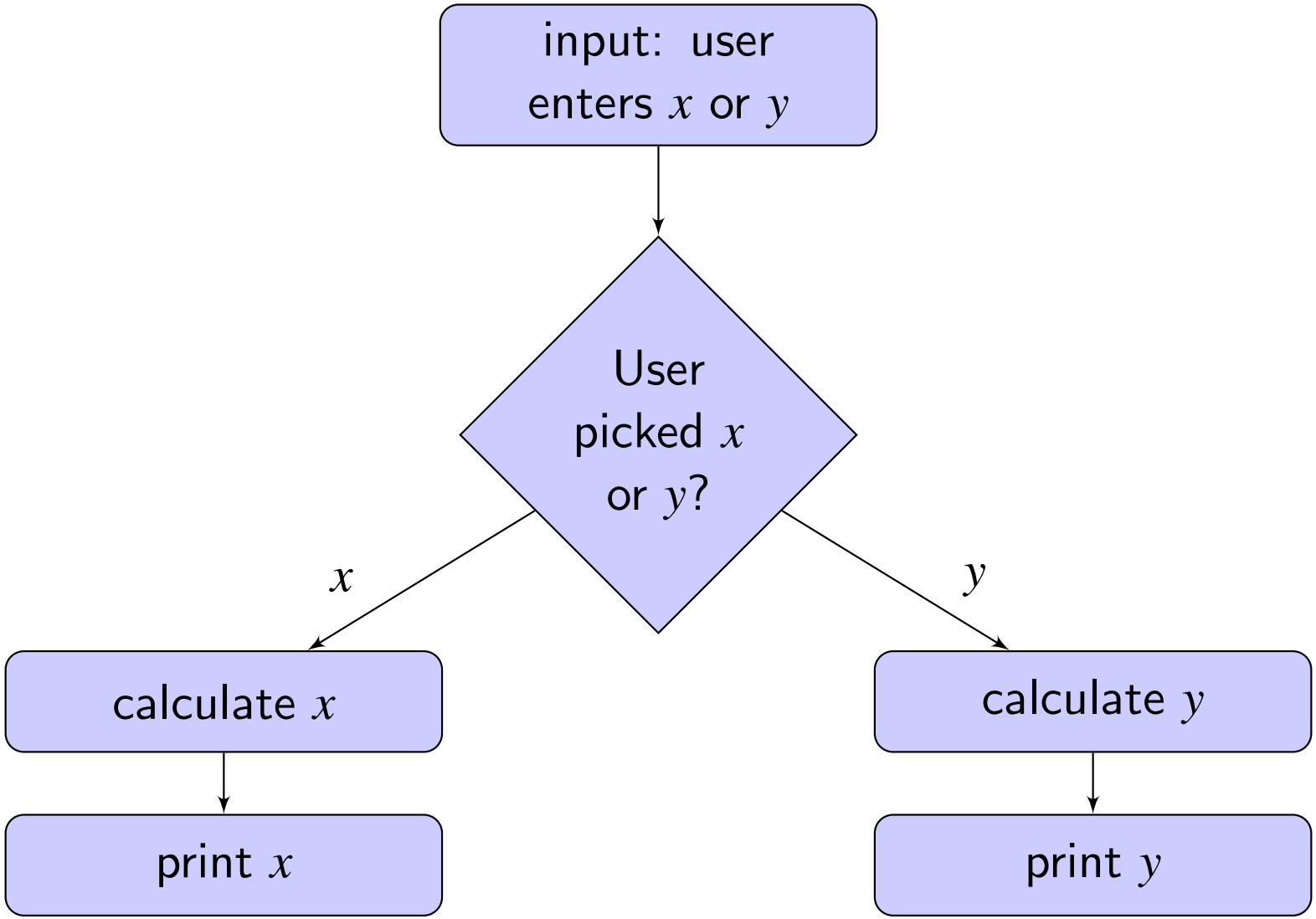
Branching Control Flow



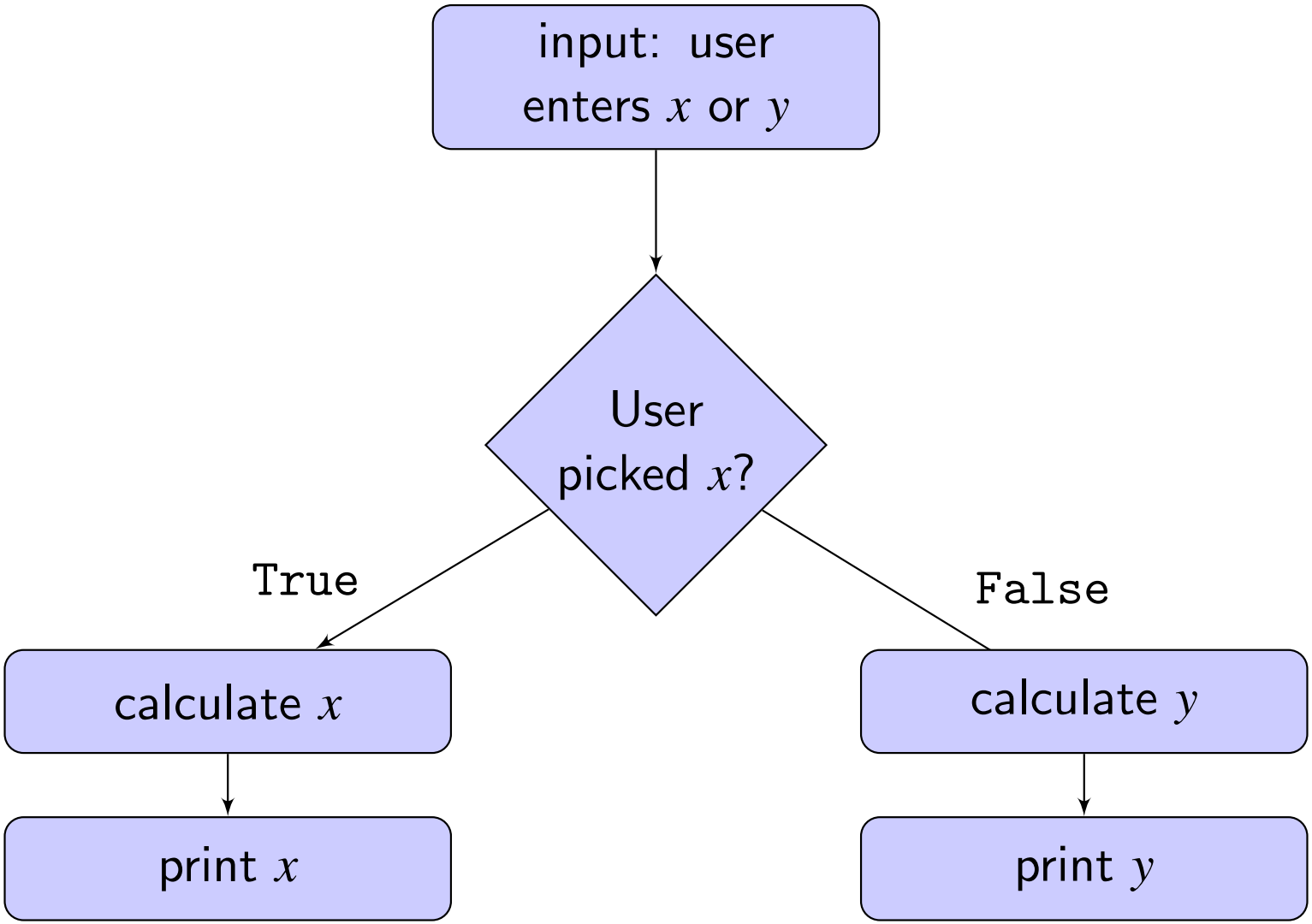
Branching Control Flow



Branching Control Flow



Branching Control Flow



bool Variables

bool Variables

The `bool` type is typically used for managing control flow.

bool Variables

The `bool` type is typically used for managing control flow.

You typically create `bool` variables using *comparison* operators.

bool Variables

The `bool` type is typically used for managing control flow.

You typically create `bool` variables using *comparison* operators.

For numeric types:

bool Variables

The `bool` type is typically used for managing control flow.

You typically create `bool` variables using *comparison* operators.

For numeric types:

- `x == y` – `True` if x equals y

bool Variables

The `bool` type is typically used for managing control flow.

You typically create `bool` variables using *comparison* operators.

For numeric types:

- `x == y` – `True` if x equals y
- `x != y` – `True` if x does *not* equal y

bool Variables

The `bool` type is typically used for managing control flow.

You typically create `bool` variables using *comparison* operators.

For numeric types:

- `x == y` – `True` if x equals y
- `x != y` – `True` if x does *not* equal y
- `x < y` – `True` if x less than y

bool Variables

The `bool` type is typically used for managing control flow.

You typically create `bool` variables using *comparison* operators.

For numeric types:

- `x == y` – `True` if x equals y
- `x != y` – `True` if x does *not* equal y
- `x < y` – `True` if x less than y
- `x <= y` – `True` if x less than *or equal to* y

bool Variables

The `bool` type is typically used for managing control flow.

You typically create `bool` variables using *comparison* operators.

For numeric types:

- `x == y` – `True` if x equals y
- `x != y` – `True` if x does *not* equal y
- `x < y` – `True` if x less than y
- `x <= y` – `True` if x less than *or equal to* y
- `x > y` – `True` if x greater than y

bool Variables

The `bool` type is typically used for managing control flow.

You typically create `bool` variables using *comparison* operators.

For numeric types:

- `x == y` – `True` if x equals y
- `x != y` – `True` if x does *not* equal y
- `x < y` – `True` if x less than y
- `x <= y` – `True` if x less than *or equal to* y
- `x > y` – `True` if x greater than y
- `x >= y` – `True` if x greater than *or equal to* y

Warning!

Warning!

We are not the same:

```
>>> x = 3
>>> x == 3
```

Warning!

We are not the same:

```
>>> x = 3
>>> x == 3
```

`x = 3` *tells* the computer: "assign the variable x the value 3"

Warning!

We are not the same:

```
>>> x = 3
>>> x == 3
```

`x = 3` *tells* the computer: "assign the variable x the value 3"

`x == 3` *asks* the computer: "is x equal to 3? Say `True` if so!"

bool Variables

bool Variables

Can also do comparisons with `str`!

bool Variables

Can also do comparisons with `str`!

Compares strings from left-to-right by ASCII codes (sort-of alphabetical order)

bool Variables

Can also do comparisons with `str`!

Compares strings from left-to-right by ASCII codes (sort-of alphabetical order)

- `str1 == str2` – `True` if `str1` and `str2` are the same

bool Variables

Can also do comparisons with `str`!

Compares strings from left-to-right by ASCII codes (sort-of alphabetical order)

- `str1 == str2` – `True` if `str1` and `str2` are the same
- `str1 != str2` – `True` if str1 and str2 are different

bool Variables

Can also do comparisons with `str`!

Compares strings from left-to-right by ASCII codes (sort-of alphabetical order)

- `str1 == str2` – `True` if `str1` and `str2` are the same
- `str1 != str2` – `True` if str1 and str2 are different
- `str1 < str2` – `True` if `str1` before `str2` in ASCII

bool Variables

Can also do comparisons with `str`!

Compares strings from left-to-right by ASCII codes (sort-of alphabetical order)

- `str1 == str2` – `True` if `str1` and `str2` are the same
- `str1 != str2` – `True` if str1 and str2 are different
- `str1 < str2` – `True` if `str1` before `str2` in ASCII
- `str1 <= str2` – `True` if `str1 < str2`, *or* `str1 == str2`

bool Variables

Can also do comparisons with `str`!

Compares strings from left-to-right by ASCII codes (sort-of alphabetical order)

- `str1 == str2` – `True` if `str1` and `str2` are the same
- `str1 != str2` – `True` if str1 and str2 are different
- `str1 < str2` – `True` if `str1` before `str2` in ASCII
- `str1 <= str2` – `True` if `str1 < str2`, *or* `str1 == str2`
- `str1 > str2` – `True` if `str1` after `str2` in ASCII

bool Variables

Can also do comparisons with `str`!

Compares strings from left-to-right by ASCII codes (sort-of alphabetical order)

- `str1 == str2` – True if `str1` and `str2` are the same
- `str1 != str2` – True if str1 and str2 are different
- `str1 < str2` – True if `str1` before `str2` in ASCII
- `str1 <= str2` – True if `str1 < str2`, *or* `str1 == str2`
- `str1 > str2` – True if `str1` after `str2` in ASCII
- `str1 >= str2` – True if `str1 > str2`, *or* `str1 == str2`

ASCII Order

ASCII Order

ASCII order is only alphabetical if everything is capitalized the same:

ASCII Order

ASCII order is only alphabetical if everything is capitalized the same:

ASCII Order

ASCII order is only alphabetical if everything is capitalized the same:

1. Special chars: Space, !, ", #, $, %, &, ’, (,), *, +, ,, -, ., /,

ASCII Order

ASCII order is only alphabetical if everything is capitalized the same:

1. Special chars: `Space`, `!`, `"`, `#`, `$`, `%`, `&`, `'`, `(`, `)`, `*`, `+`, `,`, `-`, `.`, `/`,
2. Numbers: `0`, `1`, `2`, ...,

ASCII Order

ASCII order is only alphabetical if everything is capitalized the same:

1. Special chars: `Space`, `!`, `"`, `#`, `$`, `%`, `&`, `'`, `(`, `)`, `*`, `+`, `,`, `-`, `.`, `/`,
2. Numbers: `0`, `1`, `2`, ...,
3. More special chars: `:`, `;`, `<`, `=`, `>`, `?`, `@`

ASCII Order

ASCII order is only alphabetical if everything is capitalized the same:

1. Special chars: `Space`, `!`, `"`, `#`, `$`, `%`, `&`, `'`, `(`, `)`, `*`, `+`, `,`, `-`, `.`, `/`,
2. Numbers: `0`, `1`, `2`, ...,
3. More special chars: `:`, `;`, `<`, `=`, `>`, `?`, `@`,
4. Capital letters in order: `A`, `B`, `C`, ...,

ASCII Order

ASCII order is only alphabetical if everything is capitalized the same:

1. Special chars: Space, !, ", #, $, %, &, ’, (,), *, +, ,, -, ., /,
2. Numbers: 0, 1, 2, ...,
3. More special chars: :, ;, <, =, >, ?, ,
4. Capital letters in order: A, B, C, ...,
5. More special characters: [, \,] , ^, _, ‘ ,

ASCII Order

ASCII order is only alphabetical if everything is capitalized the same:

1. Special chars: `Space`, `!`, `"`, `#`, `$`, `%`, `&`, `'`, `(`, `)`, `*`, `+`, `,`, `-`, `.`, `/`,
2. Numbers: `0`, `1`, `2`, ...,
3. More special chars: `:`, `;`, `<`, `=`, `>`, `?`, ,
4. Capital letters in order: `A`, `B`, `C`, ...,
5. More special characters: `[` , \, `]` , ^, _, ‘ ,
6. Lower-case letters in order: `a`, `b`, `c`, ...,

ASCII Order

ASCII order is only alphabetical if everything is capitalized the same:

1. Special chars: Space, !, ", #, $, %, &, ’, (,), *, +, ,, -, ., /,
2. Numbers: 0, 1, 2, ...,
3. More special chars: :, ;, <, =, >, ?, ,
4. Capital letters in order: A, B, C, ...,
5. More special characters: [, \,] , ^, _, ‘ ,
6. Lower-case letters in order: a, b, c, ...,
7. The rest of the special characters: { , |, } , ∼, DEL

ASCII Order

ASCII order is only alphabetical if everything is capitalized the same:

1. Special chars: `Space`, `!`, `"`, `#`, `$`, `%`, `&`, `'`, `(`, `)`, `*`, `+`, `,`, `-`, `.`, `/`,
2. Numbers: `0`, `1`, `2`, ...,
3. More special chars: `:`, `;`, `<`, `=`, `>`, `?`, ,
4. Capital letters in order: `A`, `B`, `C`, ...,
5. More special characters: `[` , `\`, `]` , `^`, `_`, `` ` ` ,
6. Lower-case letters in order: `a`, `b`, `c`, ...,
7. The rest of the special characters: `{` , `|`, `}` , `~`, `DEL`

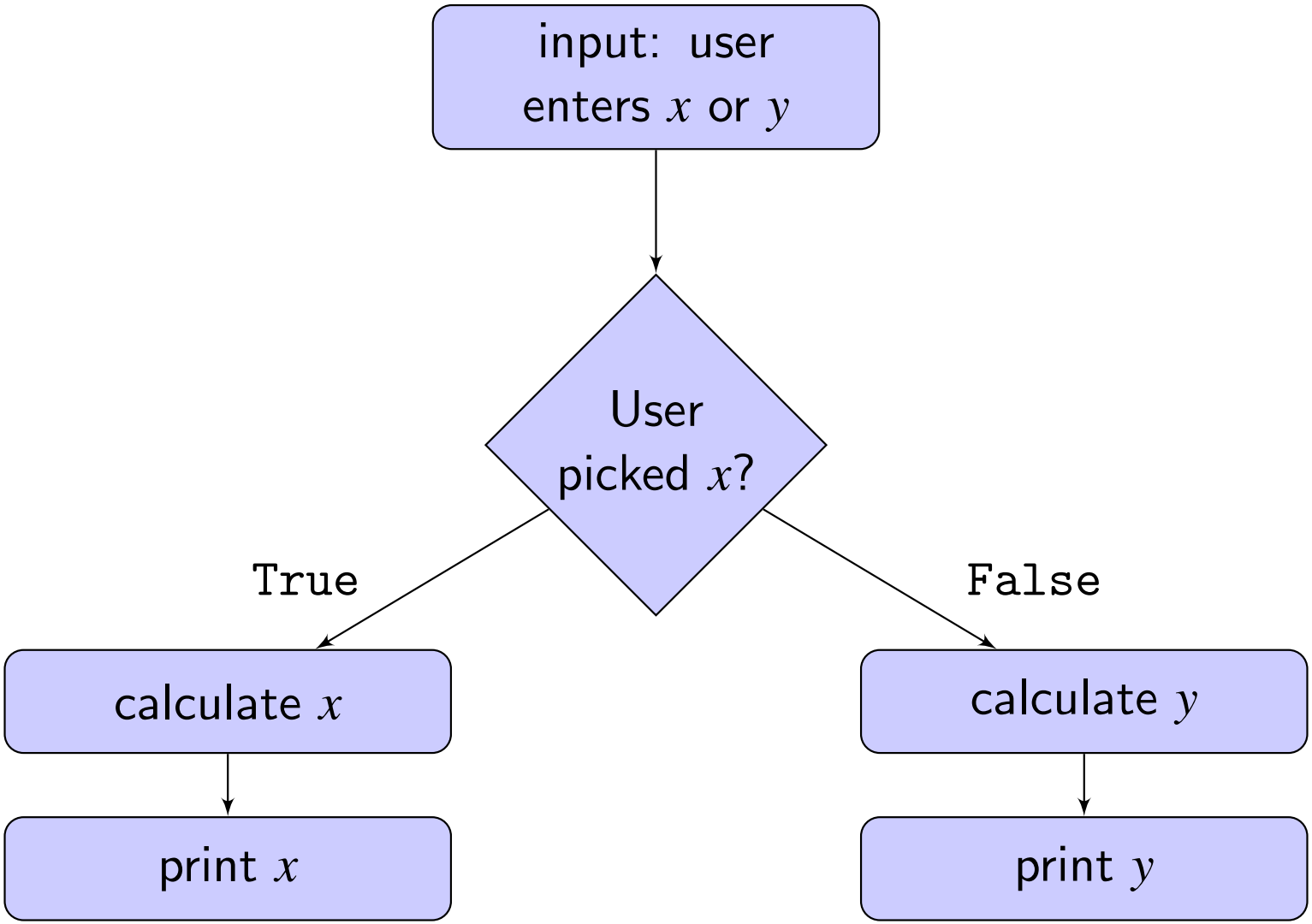
```
>>> Z < a
True
```

Concept Check!

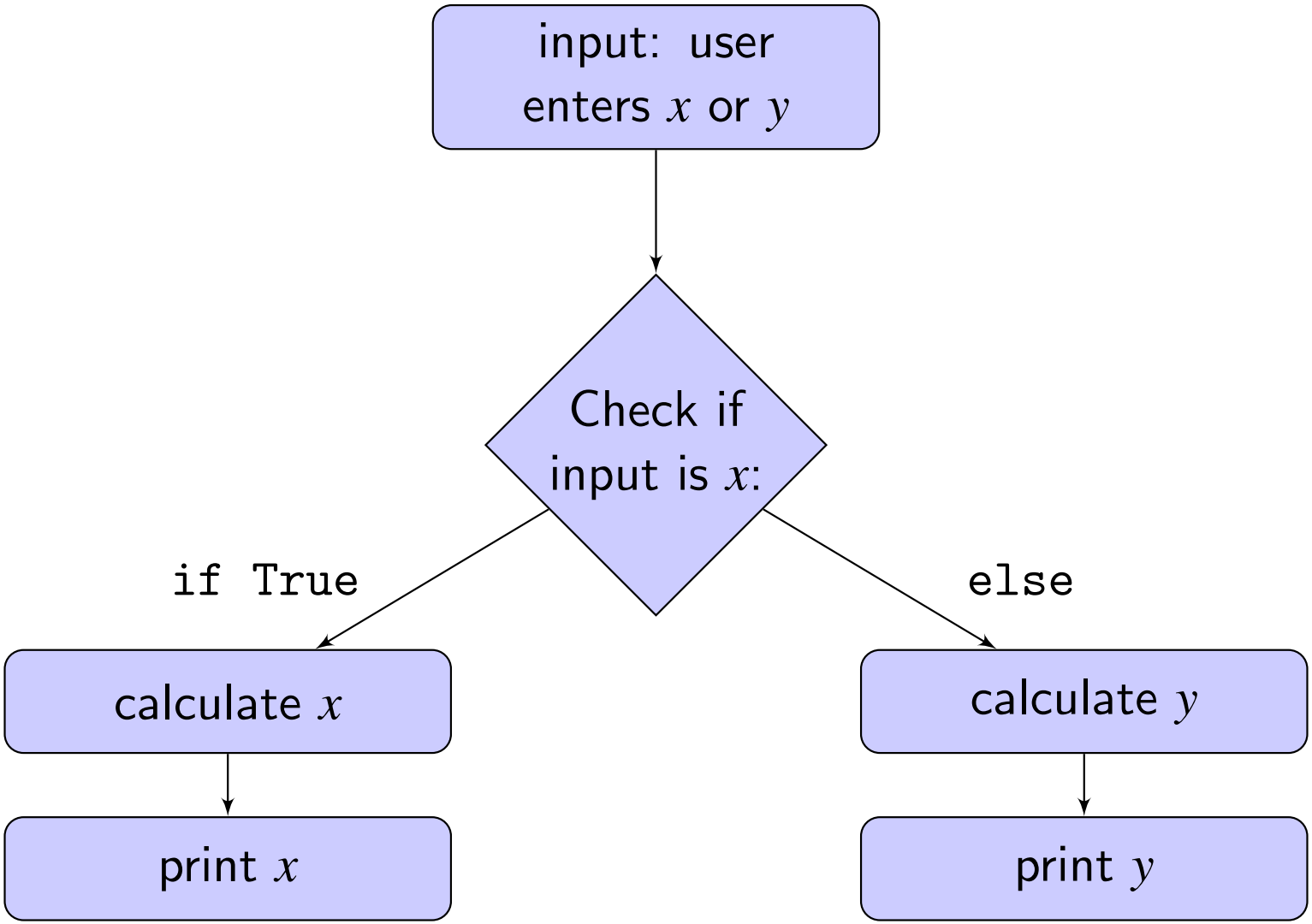
Evaluate each of the following:

1. `>>> 3 == 3`
2. `>>> 3.0 == 3.00000001`
3. `>>> 3.0 != 3.00000001`
4. `>>> 3.0 < 3.00000001`
5. `>>> 3 < 3`
6. `>>> 3 <= 3`
7. `>>> 3 > 4`
8. `>>> 3 > 3`
9. `>>> 3 >= 3`
10. `>>> "tom" < "tyler"`
11. `>>> "tom" < "Tyler"`
12. `>>> "tyler" == " tyler"`

Branching Control Flow



Branching Control Flow



if statement structure

if statement structure

```
if  condition :
```

if statement structure

`if` condition :

    This is the *body* – `print` or do something here

if statement structure

```
if condition :
    This is the body – print or do something here
    Can have more than one statement here
```

if statement structure

```
if  condition :
    This is the body – print or do something here
    Can have more than one statement here
else:
```

if statement structure

```
if  condition :
    This is the body – print or do something here
    Can have more than one statement here
else:
    print or do something else
```

if statement structure

```
if  condition :
    This is the body – print or do something here
    Can have more than one statement here
else:
    print or do something else
```

Since this is not indented, do it either way

if statement example

if statement example

```
userInput = input("Are you sick?  (enter y or n)")
```

if statement example

```
userInput = input("Are you sick?  (enter y or n)")
if userInput == "y":
```

if statement example

```
userInput = input("Are you sick?  (enter y or n)")
if userInput == "y":
    print("Stay home")
```

if statement example

```
userInput = input("Are you sick?  (enter y or n)")
if userInput == "y":
    print("Stay home")
    print("Get well soon!")
```

if statement example

```
userInput = input("Are you sick?  (enter y or n)")
if userInput == "y":
    print("Stay home")
    print("Get well soon!")
else:
```

if statement example

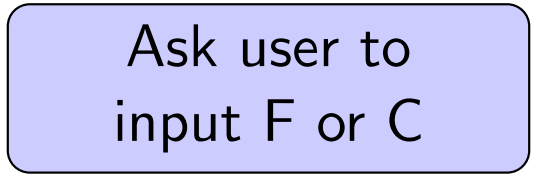
```
userInput = input("Are you sick?  (enter y or n)")
if userInput == "y":
    print("Stay home")
    print("Get well soon!")
else:
    print("You need to come to work tomorrow!")
```

if statement example

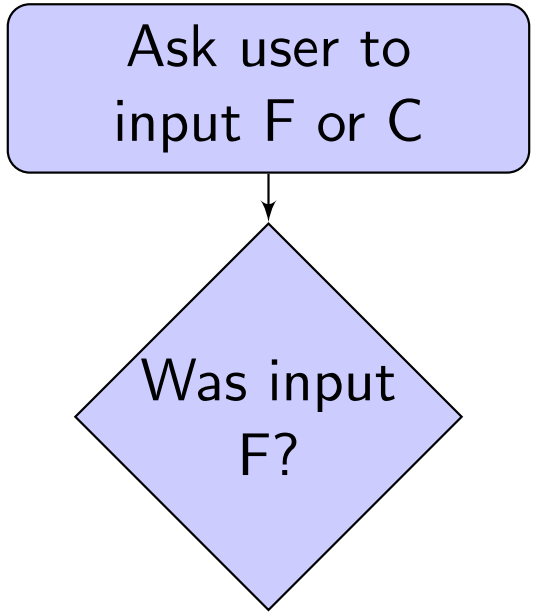
```
userInput = input("Are you sick?  (enter y or n)")
if userInput == "y":
    print("Stay home")
    print("Get well soon!")
else:
    print("You need to come to work tomorrow!")
print("Goodbye")
```

temperatures.py

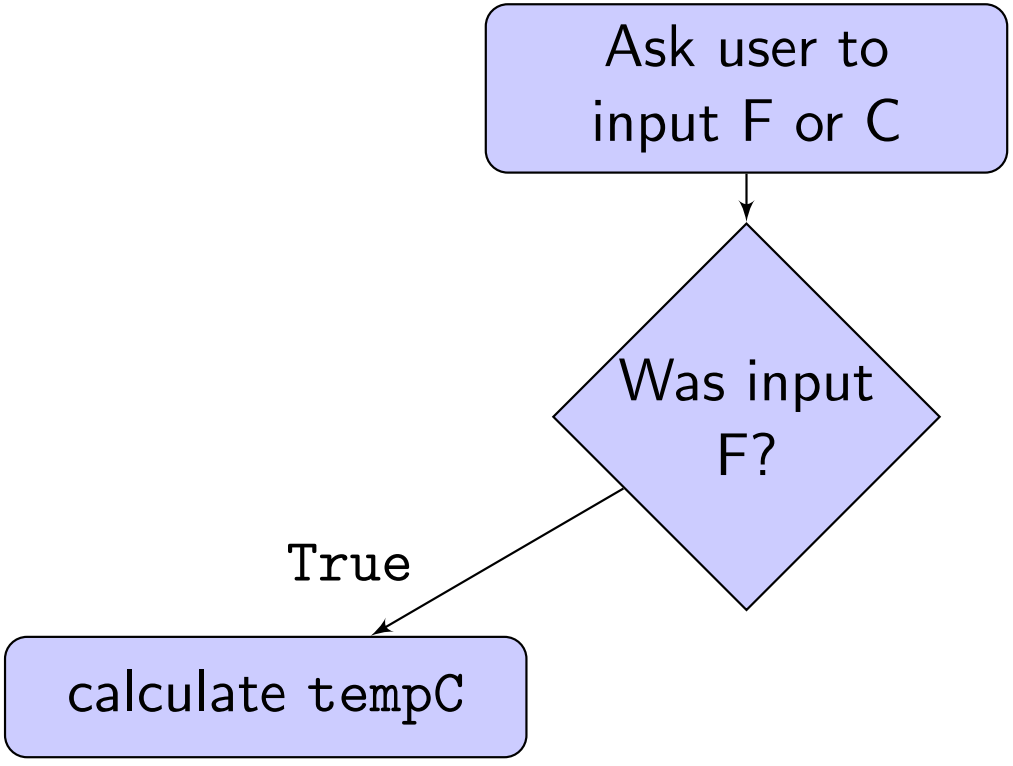
temperatures.py



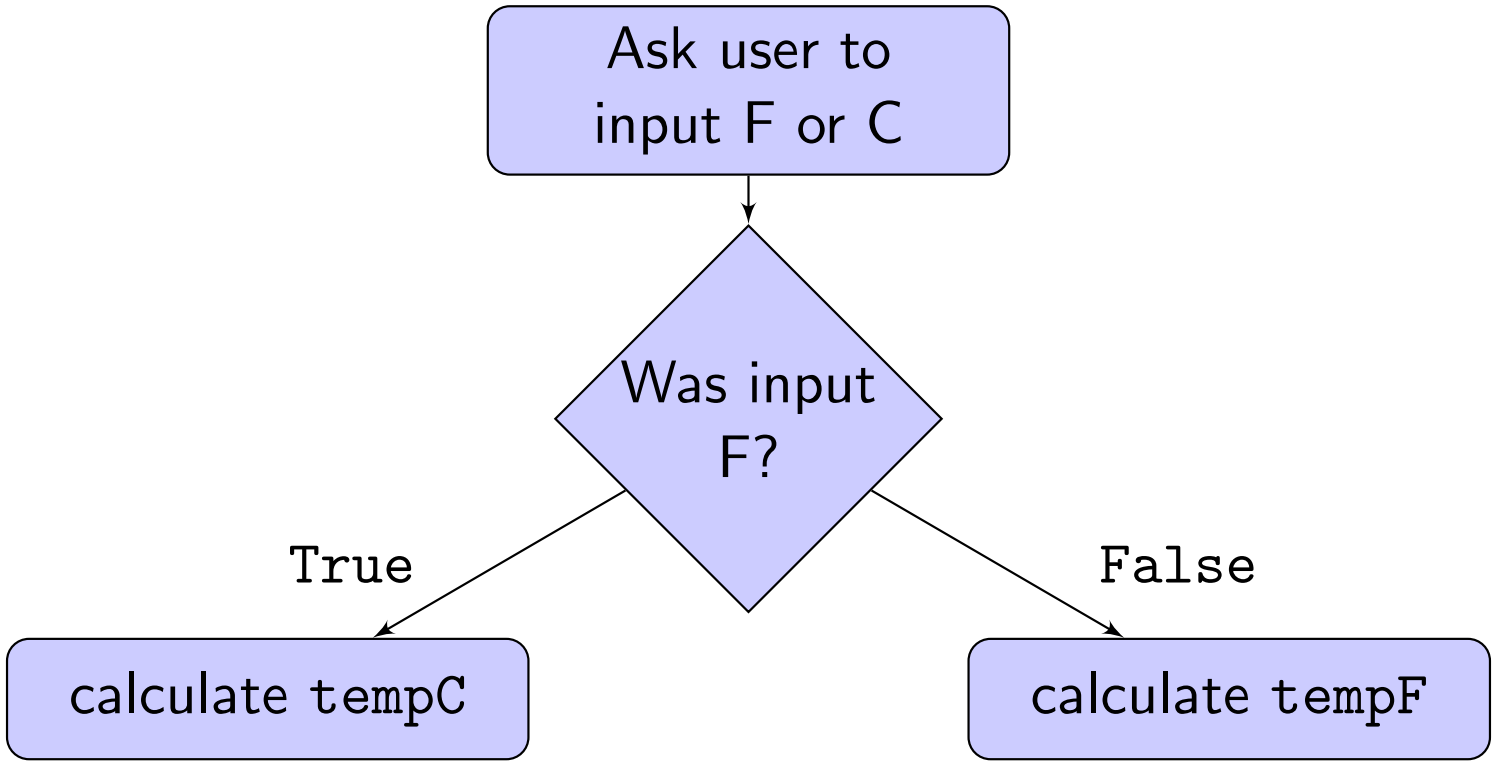
temperatures.py



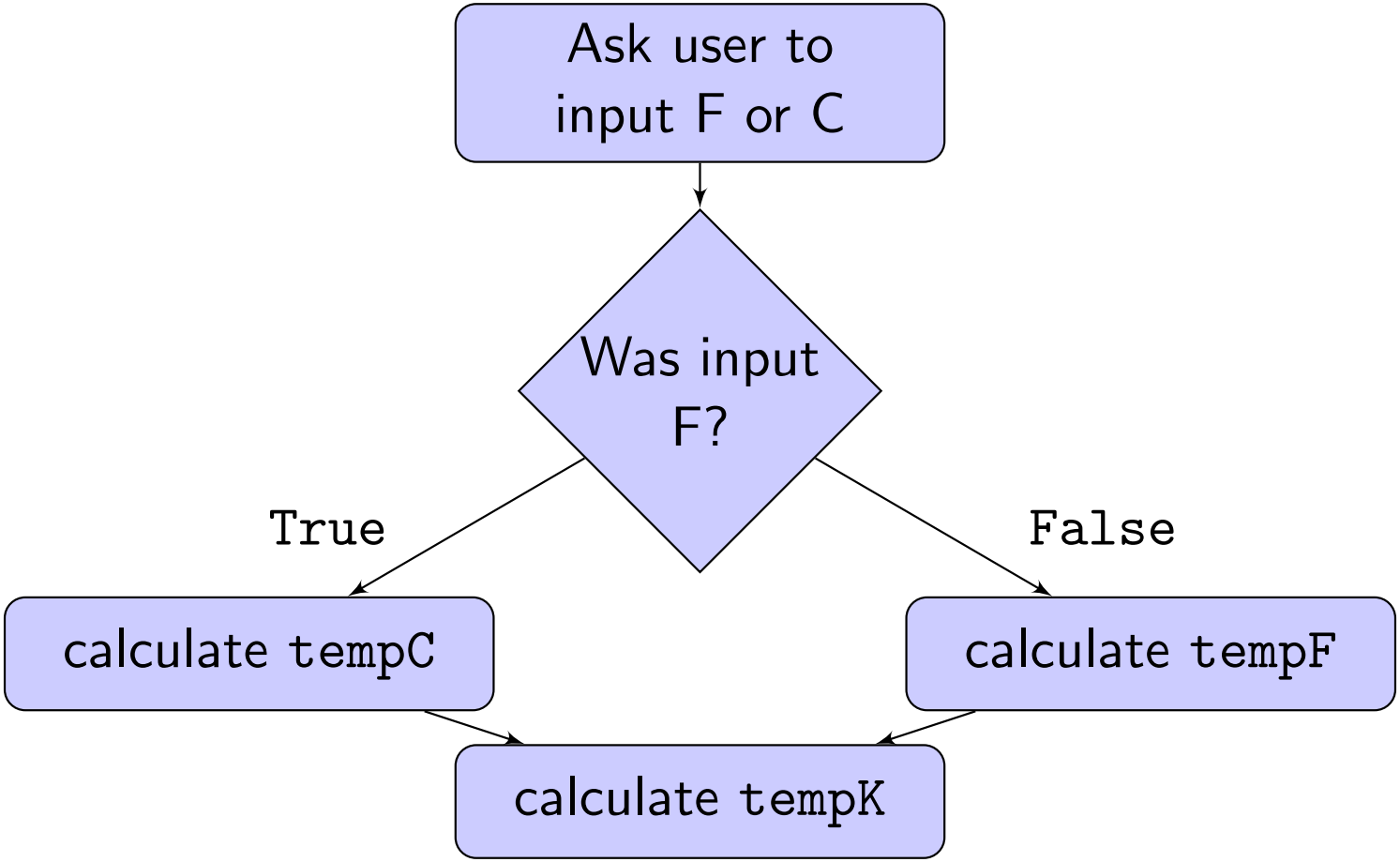
temperatures.py



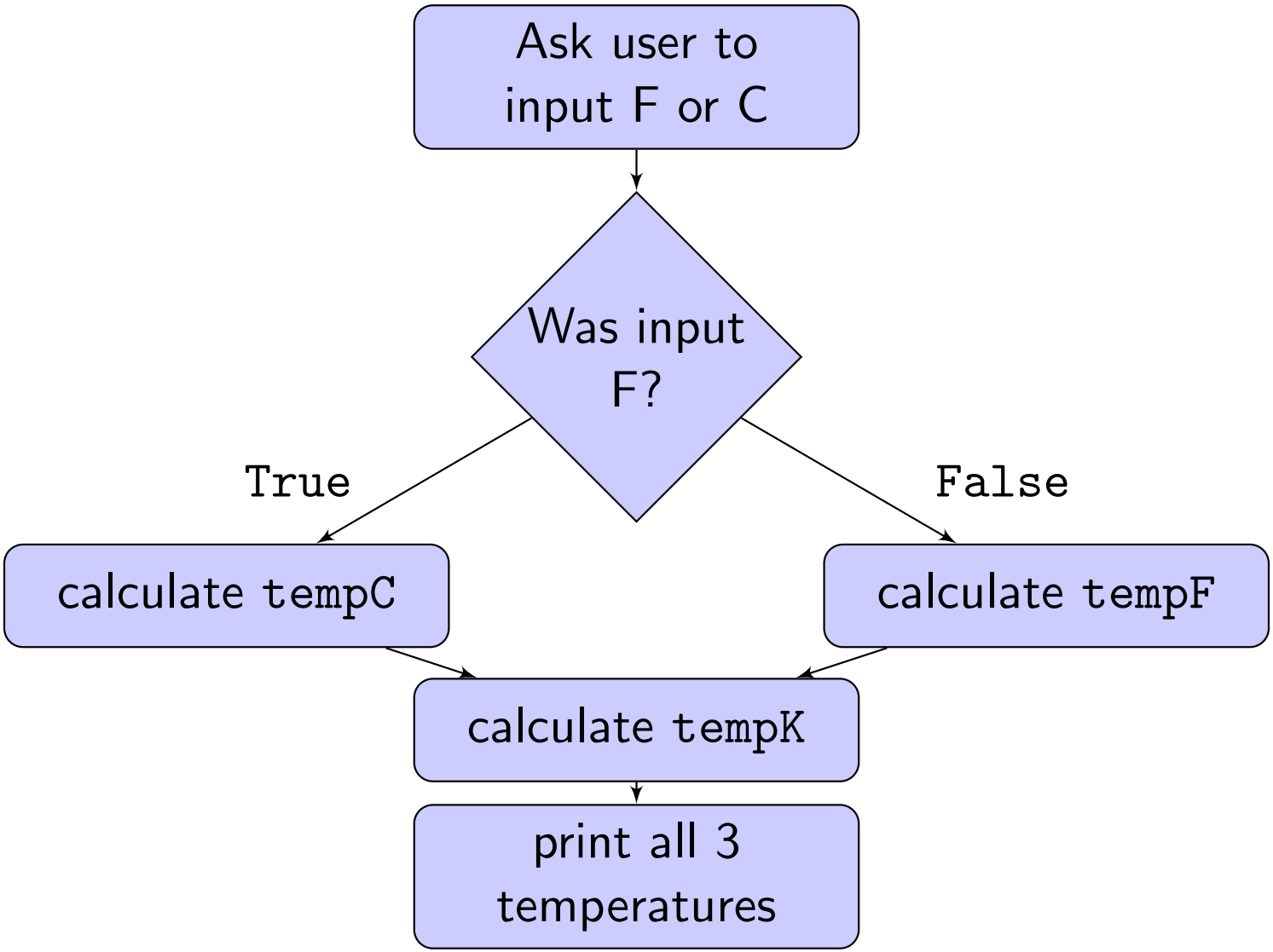
temperatures.py



temperatures.py



temperatures.py



Concept Check!

What will each of the following blocks of code print?

1. ```
   x = -5
   if x < 1.0:
       print("x is small")
   else:
       print("x is big")
   ```
2. ```
   userName = "Bobo"
   if userName >= " ":
       print("Nice name!")
   else:
       print("Bad name!")
   ```