Programming and Problem-Solving

Functions and local variables

Dennis Komm

Repetition – Control Structures

Control Flow

Control Flow – `if`

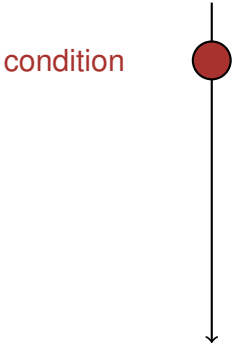
Order of the (repeated) execution of statements

- generally from top to bottom...

Control Flow – `if`

Order of the (repeated) execution of statements

- generally from top to bottom...
- ... except in selection and iteration statements

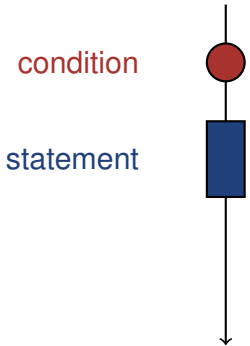



```
if condition:
    statement
```

Control Flow – `if`

Order of the (repeated) execution of statements

- generally from top to bottom...
- ... except in selection and iteration statements

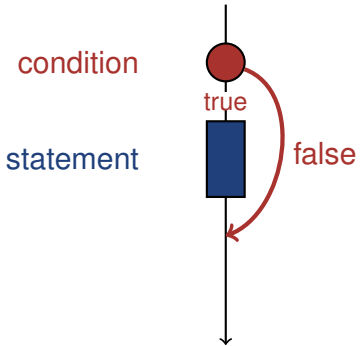



```
if condition:
    statement
```

Control Flow – `if`

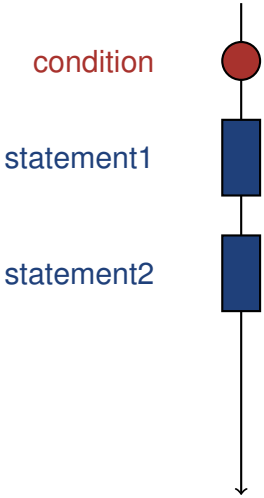
Order of the (repeated) execution of statements

- generally from top to bottom...
- ... except in selection and iteration statements



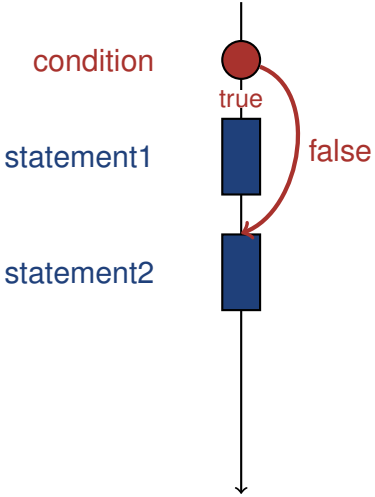

```
if condition:
    statement
```

Control Flow – `if-else`



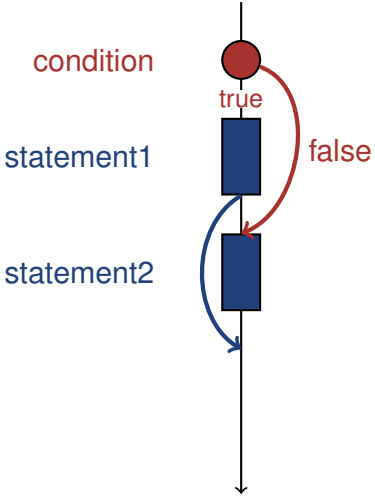

```
if condition:
    statement1
else:
    statement2
```

Control Flow – `if-else`




```
if condition:
    statement1
else:
    statement2
```

Control Flow – `if-else`

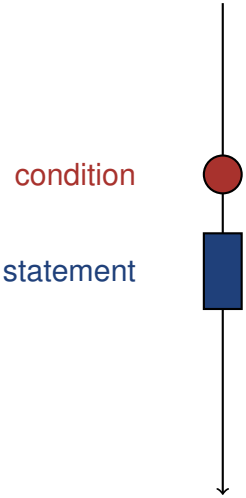



```
if condition:
    statement1
else:
    statement2
```

Control Flow – `while`

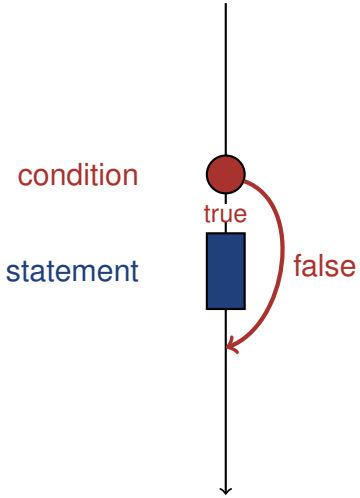
`while` *condition*:
statement

Control Flow – `while`



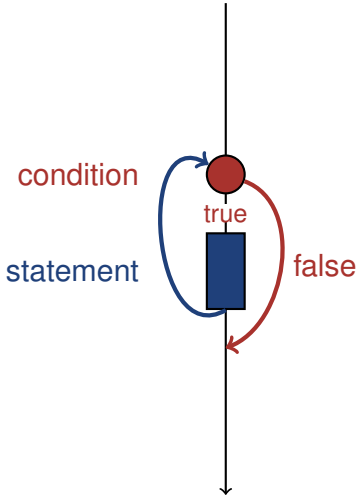
`while` *condition*:
statement

Control Flow – `while`



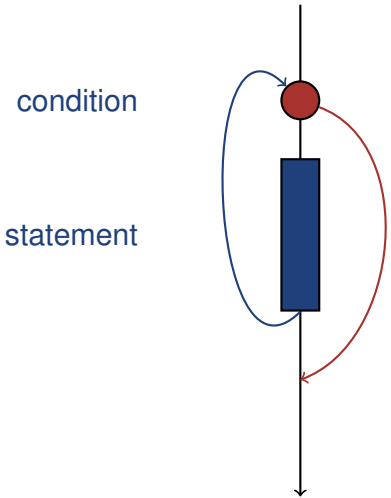
`while` *condition*:
statement

Control Flow – `while`

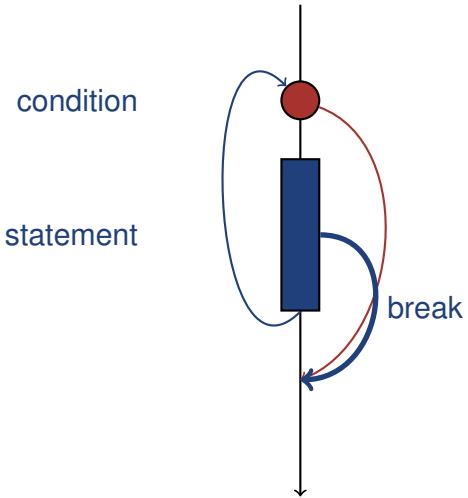


`while` *condition*:
statement

Kontrollfluss `break` in `while`-Schleife



Kontrollfluss `break` in `while`-Schleife



Functions

Functions

So far...

- One algorithm per file
- Statements are processed sequentially
- Usage of loops and control structures

Functions

So far...

- One algorithm per file
- Statements are processed sequentially
- Usage of loops and control structures

Group related code as **function**

Functions

So far...

- One algorithm per file
- Statements are processed sequentially
- Usage of loops and control structures

Group related code as **function**

```
def welcome():
    date = "March 18, 2021"
    print("Hello", username, "!")
    print("Welcome to the lecture on", date)

welcome()
```

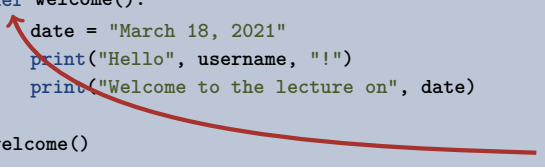
Functions

So far...

- One algorithm per file
- Statements are processed sequentially
- Usage of loops and control structures

Group related code as function

```
def welcome():
    date = "March 18, 2021"
    print("Hello", username, "!")
    print("Welcome to the lecture on", date)

welcome()
```



Functions

So far...

- One algorithm per file
- Statements are processed sequentially
- Usage of loops and control structures

Group related code as function

```
def welcome():
    date = "March 18, 2021"
    print("Hello", username, "!")
    print("Welcome to the lecture on", date)

welcome()
```

Optional list of parameters

Analogy to Natural Languages

- Python “understands” some specific words
- These are called **keywords**: `def`, `if`, `while`, etc.
- Basic stock of **functions**: `print()`, `range()`, `input()`, etc.

Analogy to Natural Languages

- Python “understands” some specific words
- These are called **keywords**: `def`, `if`, `while`, etc.
- Basic stock of **functions**: `print()`, `range()`, `input()`, etc.

`def f():` $\iff$ Python “learns” new word `f`

Analogy to Natural Languages

- Python “understands” some specific words
- These are called **keywords**: `def`, `if`, `while`, etc.
- Basic stock of **functions**: `print()`, `range()`, `input()`, etc.

`def f():` $\iff$ Python “learns” new word `f`

From Merriam-Webster dictionary

re·frig·er·a·tor

A room or appliance for keeping food or other items cool

Analogy to Natural Languages

```
  def welcome():
    date = "March 18, 2021"
    print("Hello", username, "!")
    print("Welcome to the lecture on", date)
```

```
username = input("Enter username:")
if username == "leafr" or username == "skamp" or username == "dkomm":
    welcome()
    ...
else:
    print("Username not found.")
...
```

Analogy to Natural Languages

```
  def welcome():
    date = "March 18, 2021"
    print("Hello", username, "!")
    print("Welcome to the lecture on", date)
```

```
username = input("Enter username:")
if username == "leafr" or username == "skamp" or username == "dkomm":
    welcome()
    ...
else:
    print("Username not found.")
...
```

Analogy to Natural Languages

```
  def welcome():
    date = "March 18, 2021"
    print("Hello", username, "!")
    print("Welcome to the lecture on", date)
```

```
username = input("Enter username:")
if username == "leafr" or username == "skamp" or username == "dkomm":
    welcome()


    ...
else:
    print("Username not found.")
...
```

Analogy to Natural Languages

```
  def welcome():
    date = "March 18, 2021"
    print("Hello", username, "!")
    print("Welcome to the lecture on", date)
```

```
username = input("Enter username:")
if username == "leafr" or username == "skamp" or username == "dkomm":
    date = "March 18, 2021"
    print("Hello", username, "!")
    print("Welcome to the lecture on", date)
    ...
else:
    print("Username not found.")
...
```

Analogy to Mathematical Functions

$$f(x) = 2 \cdot x + 1$$

Analogy to Mathematical Functions

$$f(x) = 2 \cdot x + 1$$

Functions in Python

- **Parameter** `x` is passed to function
- **Value** is passed back using `return`

Analogy to Mathematical Functions

$$f(x) = 2 \cdot x + 1$$

Functions in Python

- **Parameter** `x` is passed to function
- **Value** is passed back using `return`

```
def f(x):
    y = 2 * x + 1
    return y
```

Analogy to Mathematical Functions

$$f(x) = 2 \cdot x + 1$$

Functions in Python

- **Parameter** `x` is passed to function
- **Value** is passed back using `return`

```
def f(x):
    y = 2 * x + 1
    return y
```

```
def f(x):
    return 2 * x + 1
```

Analogy to Mathematical Functions

$$f(x) = 2 \cdot x + 1$$

Functions in Python

- **Parameter** `x` is passed to function
- **Value** is passed back using `return`

```
def f(x):
    y = 2 * x + 1
    return y
```

```
def f(x):
    return 2 * x + 1
```

- `return` without argument is used to simply end the function call

Analogy to Mathematical Functions

```
def f(x):
    return 2 * x + 1
```

Analogy to Mathematical Functions

```
def f(x):
    return 2 * x + 1
```

By using `return`, the function call represents the corresponding value

Analogy to Mathematical Functions

```
def f(x):
    return 2 * x + 1
```

By using `return`, the function call represents the corresponding value

- `print(f(5))` results in output `11`

Analogy to Mathematical Functions

```
def f(x):
    return 2 * x + 1
```

By using `return`, the function call represents the corresponding value

- `print(f(5))` results in output `11`
- `z = f(6)` assigns `z` the value `13`
- `z = 3 * f(2) + f(4)` assigns `z` the value `24`

Analogy to Mathematical Functions

```
def f(x):
    return 2 * x + 1
```

By using `return`, the function call represents the corresponding value

- `print(f(5))` results in output `11`
- `z = f(6)` assigns `z` the value `13`
- `z = 3 * f(2) + f(4)` assigns `z` the value `24`
- `b = (f(10) > 20)` assigns `b` the Boolean value `True`

Functions with Parameters

```
def checkuser(givenname):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if givenname in validnames:
        return True
    else:
        return False
```

```
username = input("Enter username:")

if checkuser(username) == True:
    print("Welcome", username)
    password = input("Enter your password:")
    ...
else:
    print("Username not found.")
```

Functions with Parameters

```
def checkuser(givenname):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if givenname in validnames:
        return True
    else:
        return False
```

```
username = input("Enter username:")

if checkuser(username) == True:
    print("Welcome", username)
    password = input("Enter your password:")
    ...
else:
    print("Username not found.")
```

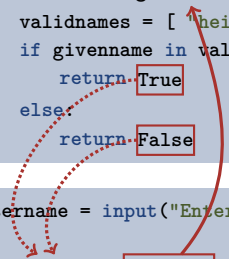
Functions with Parameters

```
def checkuser(givenname):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if givenname in validnames:
        return True
    else:
        return False
```

```
username = input("Enter username:")

if checkuser(username) == True:
    print("Welcome", username)
    password = input("Enter your password:")
    ...
else:
    print("Username not found.")
```

Functions with Parameters

```
def checkuser(givenname):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if givenname in validnames:
        return True
    else:
        return False

username = input("Enter username:")

if checkuser(username) == True:
    print("Welcome", username)
    password = input("Enter your password:")
    ...
else:
    print("Username not found.")
```



Functions with Parameters

```
username = input("Enter username:")
```

Functions with Parameters

```
username = input("Enter username:")
```

```
Enter username:
```

Functions with Parameters

```
username = input("Enter username:")
```

```
Enter username:  dkomm
```

Functions with Parameters

```
username = input("Enter username:")
```

```
Enter username:  dkomm
```

```
username = dkomm
```

Functions with Parameters

```
username = input("Enter username:")
```

```
Enter username:  dkomm
```

```
username = dkomm
```

```
if checkuser(username) == True:
```

Functions with Parameters

```
username = input("Enter username:")
```

```
Enter username:  dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

Functions with Parameters

```
username = input("Enter username:")
```

```
Enter username:  dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(givenname):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if givenname in validnames:
        return True
    else:
        return False
```

Functions with Parameters

```
username = input("Enter username:")
```

```
Enter username:  dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(dkomm):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if givenname in validnames:
        return True
    else:
        return False
```

Functions with Parameters

```
username = input("Enter username:")
```

```
Enter username:  dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(dkomm):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if dkomm in validnames:
        return True
    else:
        return False
```

Functions with Parameters

```
username = input("Enter username:")
```

```
Enter username:  dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(dkomm):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if dkomm in validnames:
        return True
    else:
        return False
```

Functions with Parameters

```
username = input("Enter username:")
```

```
Enter username:  dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(dkomm):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if dkomm in validnames:
        return True
    else:
        return False
```

```
if checkuser(dkomm) == True:
```

Functions with Parameters

```
username = input("Enter username:")
```

```
Enter username:  dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(dkomm):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if dkomm in validnames:
        return True
    else:
        return False
```

```
if False == True:
```

Functions with Parameters

```
username = input("Enter username:")
```

```
Enter username:  dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(dkomm):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if dkomm in validnames:
        return True
    else:
        return False
```

```
if False == True:
```

```
Username not found.
```

Definition of Functions

Function has to be defined **before** it can be used

Definition of Functions

Function has to be defined **before** it can be used

```
def f(x):
    return 2 * x + 1

print(f(2))
```

works, but not. . .

Definition of Functions

Function has to be defined **before** it can be used

```
def f(x):
    return 2 * x + 1

print(f(2))
```

works, but not. . .

```
print(f(2))

def f(x):
    return 2 * x + 1
```

Definition of Functions

Function has to be defined **before** it can be used

```
def f(x):
    return 2 * x + 1

print(f(2))
```

works, but not...

```
print(f(2))

def f(x):
    return 2 * x + 1
```

```
NameError: name 'f' is not defined
```

Functions

Example – Cookie Calculator

Example – Cookie Calculator

```
children = int(input("Number of children:"))
cookies = int(input("Number of cookies:"))

print("Every child receives", cookies // children, "cookies")
print("Dad receives", cookies % children, "cookies")
```

Example – Cookie Calculator

```
children = int(input("Number of children:"))
cookies = int(input("Number of cookies:"))

print("Every child receives", cookies // children, "cookies")
print("Dad receives", cookies % children, "cookies")
```

We want to make sure that `children` is positive and that each child gets at least one cookie

Cookie Calculator – Check Input

From this . . .

```
children = int(input("Number of children:"))
```

Cookie Calculator – Check Input

From this . . .

```
children = int(input("Number of children:"))
```

. . . we go to this

Cookie Calculator – Check Input

From this . . .

```
children = int(input("Number of children:"))
```

. . . we go to this

```
while True:
    children = int(input("Number of children:"))
    if children >= 1:
        break
    else:
        print("Value needs to be at least 1")
```

Cookie Calculator – Check Input

From this . . .

```
children = int(input("Number of children:"))
```

. . . we go to this

```
while True:
    children = int(input("Number of children:"))
    if children >= 1:
        break
    else:
        print("Value needs to be at least 1")
```

Analogously, we have to check that `cookies >= children`

Cookie Calculator – Getting Complicated

```
while True:
    children = int(input("Number of children:"))
    if children >= 1:
        break
    else:
        print("Value needs to be at least 1")

while True:
    cookies = int(input("Number of cookies:"))
    if cookies >= children:
        break
    else:
        print("Value needs to be at least", children)

print("Every child receives", cookies // cookies, "cookies")
print("Dad receives", cookies % children, "cookies")
```

Cookie Calculator – Getting Complicated

```
while True:
    children = int(input("Number of children:"))
    if children >= 1:
        break
    else:
        print("Value needs to be at least 1")

while True:
    cookies = int(input("Number of cookies:"))
    if cookies >= children:
        break
    else:
        print("Value needs to be at least", children)

print("Every child receives", cookies // cookies, "cookies")
print("Dad receives", cookies % children, "cookies")
```

Read and check number of children

Cookie Calculator – Getting Complicated

```
while True:
    children = int(input("Number of children:"))
    if children >= 1:
        break
    else:
        print("Value needs to be at least 1")

while True:
    cookies = int(input("Number of cookies:"))
    if cookies >= children:
        break
    else:
        print("Value needs to be at least", children)

print("Every child receives", cookies // cookies, "cookies")
print("Dad receives", cookies % children, "cookies")
```

Read and check number of cookies

Cookie Calculator – Takeaway

- The two code fragments are **nearly identical**

Cookie Calculator – Takeaway

- The two code fragments are **nearly identical**
- The following aspects are different:
  - The prompt, i.e., `"children:"` vs. `"cookies:"`
  - The minimum, i.e., `1` vs. `children`

Cookie Calculator – Takeaway

- The two code fragments are **nearly identical**
- The following aspects are different:
  - The prompt, i.e., `"children:"` vs. `"cookies:"`
  - The minimum, i.e., `1` vs. `children`
- We can outsource the code fragment into a function and thus feature **reuse**

Cookie Calculator – Takeaway

- The two code fragments are **nearly identical**
- The following aspects are different:
  - The prompt, i.e., `"children:"` vs. `"cookies:"`
  - The minimum, i.e., `1` vs. `children`
- We can outsource the code fragment into a function and thus feature **reuse**
- We have to **parameterize** the different aspects

Exercise – Cookie Calculator

Write a function that

- gets two parameters `prompt` and `minimum`
- asks the user for an integer input
- returns the input using `return` if it is at least `minimum`
- otherwise asks for a new input

Use the function in the cookie calculator

Exercise – Cookie Calculator

```
def checkinput(prompt, minimum):
    while True:
        x = int(input(prompt))
        if x >= minimum:
            return x
        else:
            print("Value needs to be at least", minimum)


children = checkinput("Number of children:", 1)
cookies = checkinput("Number of cookies:", children)

print("Every child receives", cookies // cookies, "cookies")
print("Dad receives", cookies % children, "cookies")
```

Functions

Scope and Lifetime of Variables

Local Variables

Parameters of a function are only valid within this function

```
def f(x):
    return x + 5

print(x)
```

Local Variables

Parameters of a function are only valid within this function

```
def f(x):
    return x + 5

print(x)
```

```
NameError: name 'x' is not defined
```

Local Variables

The same is true for variables that are defined in a function

```
def f(x):
    y = 5
    return x + y

print(y)
```

Local Variables

The same is true for variables that are defined in a function

```
def f(x):
    y = 5
    return x + y

print(y)
```

```
NameError: name 'y' is not defined
```

Local Variables

- Such variables (parameters) are called **local variables**
- Variables defined outside of a function are called **global variables**
- Field of validity is called **scope** of the variable
- Time in which it is defined is called **lifetime** of the variable

Local Variables

- Such variables (parameters) are called **local variables**
- Variables defined outside of a function are called **global variables**
- Field of validity is called **scope** of the variable
- Time in which it is defined is called **lifetime** of the variable

```
def f(x):
    if x < 0:
        return -100
    y = x + 1
    if y < 10:
        y += 10
    else:
        y -= 20
    return y
```

Local Variables

- Such variables (parameters) are called **local variables**
- Variables defined outside of a function are called **global variables**
- Field of validity is called **scope** of the variable
- Time in which it is defined is called **lifetime** of the variable

scope of x

```
def f(x):
    if x < 0:
        return -100
    y = x + 1
    if y < 10:
        y += 10
    else:
        y -= 20
    return y
```

Local Variables

- Such variables (parameters) are called **local variables**
- Variables defined outside of a function are called **global variables**
- Field of validity is called **scope** of the variable
- Time in which it is defined is called **lifetime** of the variable

scope of x

```
def f(x):
    if x < 0:
        return -100
    y = x + 1
    if y < 10:
        y += 10
    else:
        y -= 20
    return y
```

```
def f(x):
    if x < 0:
        return -100
    y = x + 1
    if y < 10:
        y += 10
    else:
        y -= 20
    return y
```

Local Variables

- Such variables (parameters) are called **local variables**
- Variables defined outside of a function are called **global variables**
- Field of validity is called **scope** of the variable
- Time in which it is defined is called **lifetime** of the variable

scope of x

```
def f(x):
    if x < 0:
        return -100
    y = x + 1
    if y < 10:
        y += 10
    else:
        y -= 20
    return y
```

scope of y

```
def f(x):
    if x < 0:
        return -100
    y = x + 1
    if y < 10:
        y += 10
    else:
        y -= 20
    return y
```

Global Variables

Global variables can be accessed within a function

Global Variables

Global variables can be accessed within a function

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

Global Variables

Global variables can be accessed within a function

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

Global Variables

Global variables can be accessed within a function

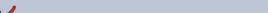
```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

local variable `y` gets value
which depends on global variable `x`

Global Variables

Global variables can be accessed within a function

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```



output global variable `x`

Global Variables

Global variables can be accessed within a function

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

Global Variables

Global variables can be accessed within a function

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

output global variable `x`

output local variable `y`

output global variable `x`

Global Variables

Global variables can be accessed within a function

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)
print(f(2))
print(x)
```

Global Variables

Global variables can be accessed within a function

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)
print(f(2))
print(x)
```

Global Variables

Global variables can be accessed within a function

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)
print(f(2))
print(x)
```

Global Variables

Global variables can be accessed within a function

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)
print(f(2))
print(x)
```

local variable z gets value that depends on global variable x and parameter y

Global Variables

Global variables can be accessed within a function

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)
print(f(2))
print(x)
```

Global Variables

Global variables can be accessed within a function

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)
print(f(2))
print(x)
```

output global variable `x`

output local variable `z`

Global Variables

Global variables can be accessed within a function

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)    <-- output global variable x
print(f(2)) <-- output local variable z
print(x)    <-- output global variable x
```

Local Variables

Local and global variables can have the same name

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

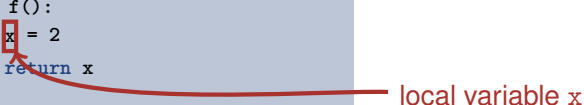
```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```



Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

output global variable x

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)      <-- output global variable x
print(f())    <-- output local variable x
print(x)
```

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)      <-- output global variable x
print(f())    <-- output local variable x
print(x)      <-- output global variable x
```

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```

parameter x gets new value that depends on its current value

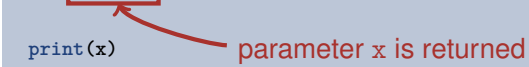
Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```



Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```

output global variable x

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)    <-- output global variable x
print(f(2)) <-- output parameter x
print(x)
```

Local Variables

Local and global variables can have the same name

- **Shadowing**
- Not forbidden, but should be avoided here whenever possible

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)      <-- output global variable x
print(f(2))   <-- output parameter x
print(x)      <-- output global variable x
```

Thanks for your attention